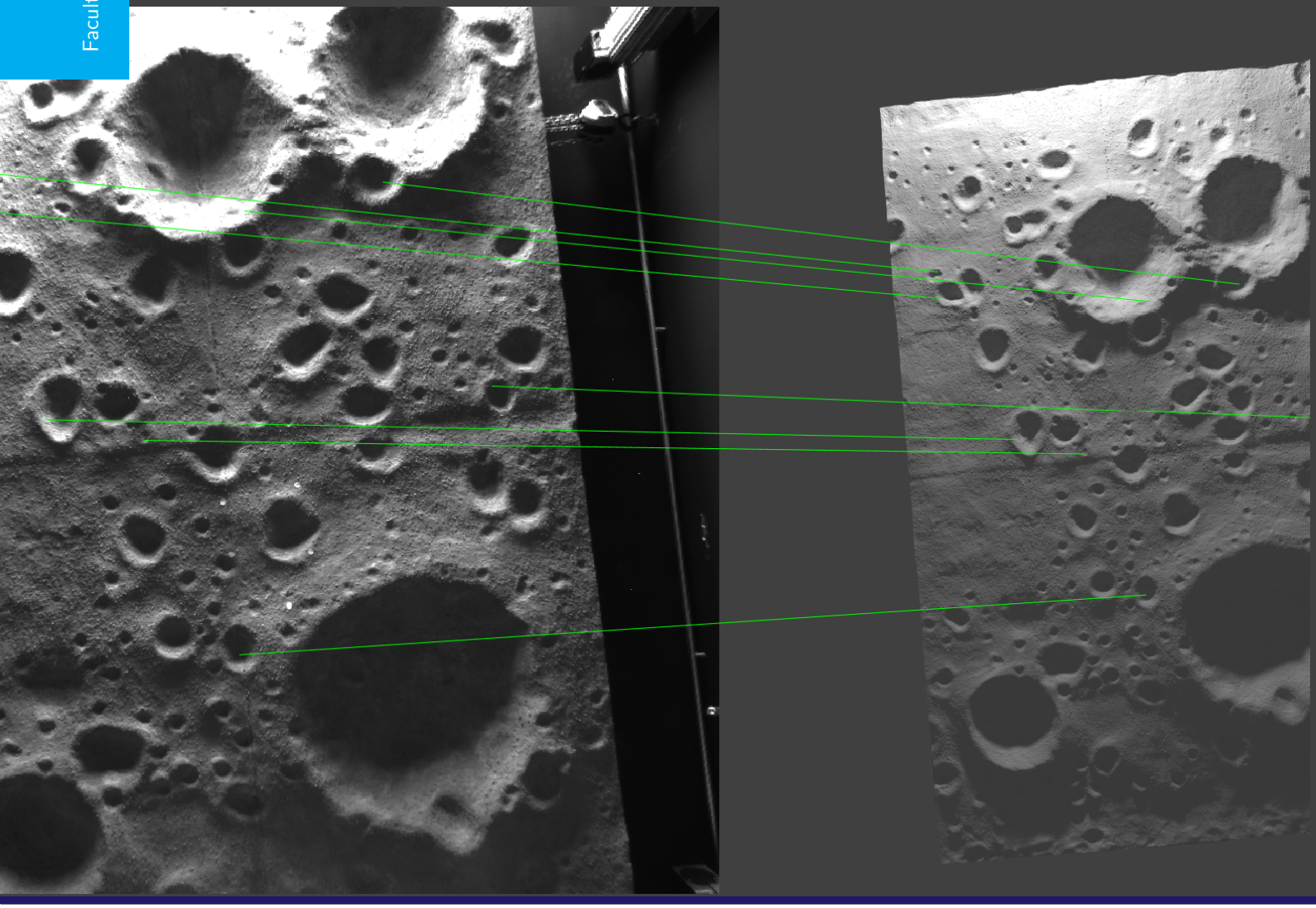
Feature-based Optical Navigation for Lunar Landings

Master Thesis

A. Beatriz Magalhães Oliveira

Faculty of Aerospace Engineering



TU Delft Delft University of Technology

Challenge the future

Feature-based Optical Navigation for Lunar Landing

Master Thesis

A. Beatriz Magalhães Oliveira

For the degree of Master of Science in Aerospace Engineering at Delft University of Technology

April 30, 2018

Student number: 4596609
Faculty of Aerospace Engineering
Delft University of Technology

Preface

Marking the end of a big and exciting chapter of my life, this report summarises the work I developed during the nearly seven months of my Master Thesis. Far from being a simple exercise in applying what I learnt so far, this journey introduced me to a completely new and unexpected field of knowledge.

Learning has always been a major part of my life and one of the most exciting things for me is connecting different fields of knowledge into a single practical application. And here, computer vision in the space exploration context. In very simple terms, teaching a lander to *see* where it is going.

Being the first time doing research, and doing so in a work environment, I've experience a lot of personal growth along the way. Having had problems communicating clearly with others and defending my own interests and opinions, depending on the input and schedules of others forced me out of my narrow comfort zone into the real world of the work place. This experience taught me to be more assertive and not to fear disagreement, while still keeping an open-mind to feedback and other perspectives.

And thanks to the laboratory work I was trusted to do partly on my own, I gained confidence in my abilities. This aspect of the thesis was both a valuable asset and the source of much complication. Not everyone gets the opportunity to see the product of their work in action with real data fresh out of the lab. The ability to validate the algorithm in such a way and to make new measurements as the research required was, indeed, very valuable. On the other hand, there is always some lab limitation that you forget to consider when preparing your commands.

For providing me the chance to work on something of importance and with possible impact to the industry, I would like to being by thanking Hans Krüger, my supervisor at DLR, and Dr. Stephan Theil. At the same level, I would like to thank my university supervisor Dr. ir. Erwin Mooij and Svenja Woicke for introducing me to the subject of optical navigation. If not for their inspiring (and fun) lectures, I wouldn't have taken the initiative to search for this opportunity at the German Aerospace Center (DLR). I would also like to thank Marco Sagliano for providing the trajectories used in the final phase of the research and the tools necessary to transform them into TRON commands.

I would like to thank, once again, Hans Krüger and professor Mooij for their time and assistance during the thesis. And finally, to all my friends for helping me clear my mind from work and avoid doing (too much) over-time.

List of Abbreviations and Notations

Glossary

AFM	Autonomous Flight Manager
AI	Artificial Intelligence
AKAZE	Accelerated-KAZE
ALHAT	Autonomous Landing and Hazard Avoidance Technology
API	Application Programming Interface
APLNav	Autonomous Precision Landing Navigation
ATON	Autonomous Terrain-based Optical Navigation
BRISK	Binary Robust Invariant Scalable Keypoints
CalDe	Camera Calibration Detection Tool
CalLab	Camera Calibration Lab
CE-3	Chang'e-3
CNav	Crater Navigation
COBALT	CoOperative Blending of Autonomous Landing Technologies
CoM	Centre of Mass
DEM	Digital Elevation Map
Descam	Descent Camera
DIMES	Descent Image Motion Estimation System
DLR	Deutsches Zentrum für Luft- und Raumfahrt (Germany Aerospace Center)
DO	Descent Orbit
DoF	Degrees of Freedom
EKF	Extended Kalman Filter
EPnP	Effective Perspective n-Point
ESA	European Space Agency
FED	Fast Explicit Diffusion
FFT	Fast Fourier Transform
FLANN	Fast Library for Approximate Nearest Neighbors
FOV	Field of View
fRPM	flipped-Radau Pseudospectral Method
GNC	Guidance Navigation and Control
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GTOC	Global Trajectory Optimization Competition
HDA	Hazard Detection and Avoidance
HDS	Hazard Detection System
IDL	Interactive Data Language
IFM	Interferometer
IMU	Inertial Measurement Unit
ISAS	Institute of Space and Astronautical Science
JAXA	Japan Aerospace eXploration Agency
JPL	Jet Propulsion Laboratory
LAlt	Laser Altimeter
LIDAR	Light Detection and Ranging
LOS	Line Of Sight
LRO	Lunar Reconnaissance Orbiter
LSQ	Least Squares
LT	Laser Tracker

LVS	Lander Vision System
MER	Mars Exploration Rovers
ML	Mapped Landmarks
M-LDB	Modified Local Difference Binary
MCMF	Moon-Centred Moon-Fixed
MSL	Mars Science Laboratory
NAC	Narrow Angle Camera
NASA	National Aeronautics and Space Administration
NDL	Navigation Doppler Lidar
NEXT	Next Exploration Science and Technology Mission
NN	Nearest Neighbour
NPAL	Navigation for Planetary Approach and Landing
OCP	Optimal-Control Problem
OF	Opportunistic Features
OpenCV	Open Source Computer Vision Library
PD	Powered Descent
PILOT	Precise Intelligent Landing using On-Board Technology
PnP	Perspective n-Point
PPL	Pinpoint Landing
QCP	Quaternion-based Characteristic Polynomial
RANSAC	Random Sample Consensus
SAPOS	Satelliten Positionierungsdienst der deutschen Landesvermessung
SHEFEX-3	SHarp Edge Flying Experiment-3
SIFT	Scale-Invariant Feature Transform
SLIM	Small Lander for Investigating Moon
SOCP	Second-Order Core Programming
SPARTAN	Shefex-3 Pseudospectral Algorithm for Reentry Trajectory Analysis
SURF	Speeded-Up Robust Features
TCP	Tool Centre Point
TIRS	Transverse Impulse Rocket System
T-MAC	Tracker-Machine
TRL	Technology Readiness Level
TRN	Terrain Relative Navigation
TRON	Testbed for Robotic Optical Navigation
VISINAV	Vision-aided Inertial Navigation
VTB	Vertical Testbed

NOTATIONS

b	Build time for FLANN index [s]
$\mathbf{C}$	Complete camera calibration matrix [-]
C_0	Projection of camera's principal point in image plane [px]
c_i	BRISK octave i, with $i \in \{0, 1, ..., n-1\}$ [-]
$\mathbf{c}_i$	Control point used in EPnP [m]
$cost$	Value of FLANN index optimisation cost function [-]
$\mathbf{c}^X$	Checkerboard corner point in the X reference frame [m]
d	Distance from camera to world point [m]
$\mathbf{d}_c$	Vector pointing to world point in the camera reference system [m]
d_i	BRISK intra-octave i, with $i \in \{0, 1, ..., n-1\}$ [-]
$\mathbf{d}_p$	Vector pointing to image point in the homogeneous image pixel coordinate system [px]
d_{query}	Distance of query point to its matched data point [m]
$\mathbf{d}^X$	Direction vector pointing from the origin of $\mathcal{F}_C$ in the direction of the world point in $\mathcal{F}_X$ [m]
$\mathbf{d}_S^X$	Scaled direction vector pointing from the origin of $\mathcal{F}_C$ to the world point in $\mathcal{F}_X$ [m]
$\mathbf{E}$	Extrinsic parameters matrix [-]
f	Focal length [m]

f_u	Horizontal magnification [px]
f_v	Vertical magnification [px]
h	Altitude [m]
h	Number of segments used to divide time period in hp pseudospectral methods to solve OCP [-]
$\mathbf{H}$	Hessian matrix [-]
h_{surf}	Surface altitude [m]
$\mathbf{I}$	Identity matrix [-]
J	Cost function in Bloza problem [-]
$\mathbf{K}$	Intrinsic parameters matrix [-]
K	Branching factor of hierarchical K-means tree [-]
k_x	xth order radial distortion parameter [-]
$K_{x,y}$	Entry of $\mathbf{K}$ at row x and column y [px]/[-]
L	Number of elements in $\mathscr{L}$ [-]
L	Limit number of searches in a hierarchical K-means tree [-]
$L^i_{Hessian}$	Discriminant of image i [-]
L^i_{jk}	Second order derivates of image i of the scale-space used in AKAZE [-]
m	Memory load for FLANN index [-]
$\mathbf{M}$	Matrix representing the linear equations used in EPnP [-]
m_u	Horizontal pixel dimension [m/px]
m_v	Vertical pixel dimension [m/px]
n	Number of BRISK octaves and intra-octaves [-]
n_{div}	Number of rows and colums to sub-divide image during feature pre-selection [-]
$n_{f/div}$	Number of features per image subdivision [-]
n_{lim}	Limit of features in pre-selection [-]
O	Number of AKAZE octaves [-]
o	AKAZE octave number [-]
O^Y_X	Origin of $\mathscr{F}_X$ in the $\mathscr{F}_Y$ reference frame [m]
p	Number of discrete nodes in pseudospectral methods to solve OCP [-]
par_i	Grid search parameter i [-]
$\mathbf{p}_i$	Reference point used in EPnP [m]
$\mathbf{P}_i$	Image point; projection of world point, $\mathbf{P}_W$ onto image plane [px]
$\mathbf{p}_{query}$	Query point [m]
$\mathbf{P}_W$	World point [m]
p_x	xth order tangential distortion parameter [-]
$\mathbf{p}^X$	Position vector in $\mathscr{F}_X$ [m]/[px]
$\mathbf{q}$	Quaternion [-]
q_0	Real part of quaternion vector [-]
q_i	Imaginary entry of quaternion vector, where $i \in \{1,2,3\}$ [-]
q'	Quaternion vector in the form used by SensorDTM [-]
q_x	Entry x of quaternion $\mathbf{q}$; q_0 is scalar element [-]
r	Radial distance [m]
$\mathbf{R}^B_A$	Rotation matrix from $\mathscr{F}_A$ to $\mathscr{F}_B$
$R_{i,j}$	Entry at line i and column j of $\mathbf{R}$ [-]
r_{Moon}	Lunar radius [m]
S	Number of AKAZE sub-levels [-]
s	AKAZE sub-level number [-]
s	Saliency score used in BRISK [-]
s	Search time for FLANN index [s]
$\mathbf{T}$	Translation vector [m]
$\mathbf{u}$	Control vector in Bloza problem [-]
(u,v)	Image coordinates [px]
(u_0,v_0)	Pixel coordinates of C_0 in $\mathscr{F}_i$ [px]
w_i	Quocient used for the option i of the quaternion calculation from a rotation matrix, where $i \in \{1,2,3,4\}$ [-]
w_{max}	Maximum quocient for the quaternion calculation [-]

w_b	Weight factor for tree build time versus search time for FLANN index optimisation [-]
w_m	Weight factor for memory load versus computational time load for FLANN index optimisation [-]
$\mathbf{x}$	State vector in Bloza problem [-]
$\hat{\mathbf{x}}_A^B$	Unit vector of X_A in $\mathscr{F}_B$ coordinates
(x, y, z)	Cartesian coordinates in MCMF reference frame [m]
(x_C^X, y_C^X, z_C^X)	Camera's coordinates in $\mathscr{F}_X$ [m]
(x_S^X, y_S^X, z_S^X)	Lamp's or Sun's coordinates in $\mathscr{F}_X$ [m]
(x_i, y_i, z_i)	Image point coordinates [px]
(x_W, y_W, z_W)	World point coordinates [m]
(x^X, y^X, z^X)	Coordinates in $\mathscr{F}_X$ [m]/[px]
$\hat{\mathbf{y}}_A^B$	Unit vector of Y_A in $\mathscr{F}_B$ coordinates
$\hat{\mathbf{z}}_A^B$	Unit vector of Z_A in $\mathscr{F}_B$ coordinates
$\mathscr{L}$	Set of long-distance sampling pairs used in BRISK [-]
$\mathscr{S}$	Set of short-distance sampling pairs used in BRISK [-]
α_{ij}	Parameters used to describe $\mathbf{p}_i$ as a linear combination of $\mathbf{c}_j$ [-]
δ	Latitude [rad]
δ_X	Displacement of pixel in X-direction [px]
ϵ	Camera exposure [s] / lamp strength [-]
η	Depth data (z-depth [Blender unit] or z-value [m])
σ	Scale of image [-]
$\sigma_{i,norm}$	Normalised scale factor of image i of the scale-space used in AKAZE [-]
τ	Longitude [rad]
τ'	Longitude in CE-3 data form [rad]
θ	Angle between the X_i- and Y_i-axis [rad]

Reference Systems

$\mathscr{F}_C$	Camera reference frame
$\mathscr{F}_C'$	Camera reference frame used by rendering softwares ($Y_{C'}$ and $Z_{C'}$ point in direction oposite from Y_C and Z_C)
$\mathscr{F}_i$	image reference frame
$\mathscr{F}_{KUKA}$	KUKA world reference frame
$\mathscr{F}_{PHY}$	Terrain model 3 reference frame centred at the corner of the model
$\mathscr{F}_{LT}$	laser tracker reference frame
$\mathscr{F}_M$	Model reference frame
$\mathscr{F}_{MCMF}$	Moon-Centred Moon-fixed reference frame
$\mathscr{F}_{TMAC}$	TMAC reference frame
$\mathscr{F}_{TRON}$	TRON reference frame
$\mathscr{F}_z$	depth data reference frame
X_A	X-axis of $\mathscr{F}_A$ reference frame
Y_A	Y-axis of $\mathscr{F}_A$ reference frame
Z_A	Z-axis of $\mathscr{F}_A$ reference frame

Contents

1 Introduction

This introduction begins with a brief non-comprehensive overview of the state of the art for Pinpoint Landing (PPL) (Section 1.1). In Section 1.2, the research questions are presented along with the navigation concept studied in this thesis. Finally, the structure of the report is outlined in Section 1.3.

1.1. State of the Art

When planning for a planetary landing, the landing site plays a crucial role. On one hand, there is usually a scientific objective behind the mission. On the other hand, there are technological limitations.

These two sides tend to clash, since the scientific interest is often concentrated on or near hazardous areas. Mountains and craters can have steep slopes, which may cause the lander to topple over. Rocky terrain and canyons can reveal much about the geological history of a celestial body, but landing on a rock or falling through a ravine would lead to a mission failure.

Thus, to avoid damaging the lander on touchdown, the nominal landing site is selected so that the associated landing ellipse is within an inherently safe area. A landing ellipse is the area on the surface determined through analysis (*e.g.*, a Monte-Carlo simulation) where the lander is expected to land 99% of the times. An area is considered inherently safe if it is devoid of hazards, such as steep slopes and rocks (the dimensions of which depend on the lander itself).

For reference, the ellipse size for the Curiosity rover was 9.5×3.5 km [J. Erickson and J.P. Grotzinger, 2014]. When comparing with prior missions, this ellipse is quite small. Still, it still imposes significant constraints on the possible landing sites.

To address this issue, two strategies can be explored: reduce the susceptibility to hazards or the size of the landing ellipses. For the same landing ellipse, the stricter the requirements for an area to be considered safe, the less locations can be found to meet them. An example of a concept that takes the first approach is Hazard Detection and Avoidance (HDA). A lander equiped with HDA is capable of recognising the hazards and actively searches for and steers into safe landing spots.

On the other hand, reducing the size of the landing ellipse can be done by improving the Guidance Navigation and Control (GNC) system, so that the lander can remain closer to the nominal trajectory and, as a consequence, to the target landing site. Examples of limiting factors in this case are the model used for the environment (*e.g.*, gravity models), the one used for the vehicle's dynamics and the accuracy of the navigation solution during the flight.

Both of these strategies can be implemented in the same mission and can be used to achieve PPL (defined as landing within 100 m of a planned location).

The only lander to date that was able to achieve autonomous PPL was the Chinese Chang'e-3 (CE-3) lunar lander [Li et al., 2015]. The final landing was within about 89 m of the planned landing spot. The significance of this achievement is clear when noting that previous lunar landings had landing errors in the order of kilometres. Li et al. [2015] credit the success of the mission to the autonomy of the GNC system used during Powered Descent (PD) and soft landing, and the corrections performed to the Inertial Measurement Unit (IMU) measurements during this period.

Currently, both National Aeronautics and Space Administration (NASA) and European Space Agency (ESA) are developing new solutions to the PPL problem.

Autonomous Landing and Hazard Avoidance Technology (ALHAT) is NASA's project set to achieve autonomous and safe precision landing on any solid body in the solar system, regardless of lighting conditions [Carson et al., 2014a]. Part of this system includes Terrain Relative Navigation (TRN) – the ability to determine the lander's state with respect to the target surface. This property has been recognised as highly valuable, since it can be used to perform PPL and to avoid large scale landing hazards (thus increasing the selection of landing sites with high scientific value). For these reasons, it will be implemented in the Mars 2020 lander mission.

In this context, various approaches are being considered. The first innovation under study is based on the long range altimeter described by Pierrottet et al. [2014]. The instrument can be used, starting from a distance of 30 km, to create a height map of the underlying surface. This map can then be used for TRN by matching it with an on-board Digital Elevation Map (DEM). The accuracy achieved through this method is of around 90 m

provided an initial position within 1.6 km of the real one [Johnson and Ivanov, 2011].

The second approach is based on optical data captured through a monocular camera [Johnson et al., 2015]. Once again an on-board map and an initial pose estimate is required. In this case, the on-board map is an image representing the target surface, namely the full landing ellipse (for a Mars mission) or the surface covered through the descent orbit and PD (for a lunar mission).

The estimated state is used to rectify the camera image; in other words, the image is warped to into the perspective of the onboard map. The same state is also used to crop a section of a low resolution map, which is too large to use as is. In an initial coarse iteration, the rectified image is matched to the cropped map.

The resulting solution is then refined. First, it is used to crop a second smaller section of the full resolution map. From the camera image, up to 100 small image patches are extracted and matched to the map through spatial correlation. With a total of nine to ten images processed in ten seconds fused with IMU measurements through an Extended Kalman Filter (EKF), the pose estimation is reduced to the order of tens of metres.

An initial state estimate is required within 3 km error for position and 0.15° for attitude. This approach was successfully tested in 2014 [Trawny et al., 2015].

In either case, NASA's approach is to improve state measurements obtained through IMU and Star Trackers, during descent orbit and PD, using TRN (via laser altimeter data or visual match results).

ESA's project dedicated to this endeavour, first named Next Exploration Science and Technology Mission (NEXT) and later renamed Lunar Lander was canceled in 2012. Nevertheless, part of the research on GNC continues. Along with the Russian Space Agency, ESA is developing the Precise Intelligent Landing using On-Board Technology (PILOT) system, which will include the ability to perform PPL. PILOT is to be included in later Russian Moon landers.

Requiring, once again, an initial estimate of the lander's state, ESA's approach is based on crater-detections in camera images [Mammarella et al., 2011]. These craters are matched to an on-board database obtained from geo-referenced DEMs and images. The matching is performed by first transforming the database crater coordinates into the image space, using the state estimate. The craters in the image are detected and the two point clouds are matched using a Nearest Neighbour (NN) search. This method requires a precision of 0.1° for the initial attitude estimation, and is to be applied before the beginning of the PD.

1.2. Research Questions

Recognising the need for high precision navigation solutions to achieve PPL and the reliance of current methods on initial state estimates, the following **research question** arose:

What navigation precision can be achieved with an autonomous optical navigation system for lunar landings without an initial state estimate?

The landing accuracy itself (which defines PPL) is highly dependent on the guidance and control systems used; thus, it is affected by the lander's dynamics and its interaction with the environment. In other words, the impact of an optical-based method for navigation in the landing accuracy can only be evaluated for a given vehicle and corresponding guidance and control systems, along with any additional sensors (such as IMUs and Sun sensors).

By characterising the proposed navigation algorithm in terms of the individual navigation solutions, the result is not limited to one lander and one guidance and control system. Instead, the expected navigation accuracy would hold, regardless of the mission. Along with an estimated order of magnitude for the solution frequency, these parameters can be compared to mission requirements or considered when integrating it into a GNC system (*e.g.*, by fusing it with IMU measurements).

For these reasons, the focus of this research is the development of a navigation algorithm that does not require an initial guess of the lander's state; the evaluation of its performance in terms of navigation accuracy and an estimate of the required time per solution.

The proposed algorithm will be based on optical camera images. Whereas the NASA approach uses image correlation, and ESA uses crater detection, the concept here explored is based on generic image features (explained in Chapter 3). The camera images captured in flight are used to detect features, which are then matched to a database.

This database is generated offline. Doing so requires the planned nominal trajectory for the lander and a DEM of the target surface. Using a 3D model of the surface (based on the DEM) and the expected illumination conditions (which, for a lunar landing, can be precisely determined for a given instance in time), a virtual camera can be positioned at points of the nominal trajectory. Images are rendered at these points, and features

are extracted from them. The 3D coordinates associated with each feature and its feature descriptor – vector representative of the appearance of a feature (explained in Chapter 3) – are used to build the database.

During the flight, once a set of matches has been found between a captured (online) image and the database, the state of the camera (and, as a consequence, the state of the lander) can be determined.

A dependency with the quality of available data is expected, namely on the accuracy of the DEMs, of the trajectory and illumination conditions. As such, we can define the theoretical potential of the algorithm as being the expected performance when using ideal data (high resolution DEM, exact correspondence between nominal and flown trajectory and known illumination conditions). Studying this potential is important since the quality of available data is continuously improving thanks to various exploration efforts. Still, this will not take into account possible future improvements of the image rendering software available (required to generate the on-board database).

On the other hand, it is also important to contrast this potential with what can currently be achieved, *i.e.*, using available DEMs and incomplete or imprecise trajectory and illumination conditions. This last uncertainty is only of importance when dealing with small bodies such as asteroids.

In accordance, the following **sub-questions** were derived:

1. *How accurate is the navigation solution when using ideal data?*
2. *How sensitive is the navigation solution to deviations between the offline data used to generate the database and the real online conditions?*
3. *How representative are the offline images generated through currently available software and actual DEMs?*

A rendered image will be considered representative of the corresponding real image if at least 80% of the N best matches between the two are true matches (this measure was based on the characteristics of the method used for pose determination, discussed in Chapter 4.4). N is the number of matches to be used for pose determination and is one of the algorithm parameters to be determined in Chapter 7.1. The offline data used to generate a database include the nominal trajectory and illumination conditions.

The accuracy of the navigation solution will be measured through the norm of the position error vector, when tuning the algorithm's parameters (Chapter 7.1). Using a single value facilitates the analysis. For a more precise characterisation, the error in each of the Cartesian directions and the angle between the real and calculated camera frame vectors will be considered. The position errors will be considered as a percentage of the real Line Of Sight (LOS) distance to the surface.

For time considerations, only an estimate of the order of magnitude can be expected. This is a consequence of the inability to accurately measure computation time: the available methods only measure the clock time, meaning that it is affected by other parallel processes and is dependent on the computer itself. Additionally, the software to be developed will not be entirely optimised. It is recognised that using different algorithms for the same task can drastically change the computation time required (as was observed during the parameter study in Section 7.1.1). The optimisation of the software is beyond the scope of this research.

The trajectories will be generated using an open-loop optimiser – it will not take into account the navigation solutions derived from the algorithm. The non-nominal (disturbed) trajectories will be obtained by changing the initial conditions. The final landing spot will be the same in all cases.

Taking all the above into account, the **goals** derived are:

1. Design the software and associated work-flow for the database generation (offline script) and for the pose determination (online script)
2. Analyse the theoretical potential of the navigation software
   - Determine the navigation accuracy when using ideal data and real online images
   - Determine the changes in navigation accuracy caused by deviations between the nominal and flown trajectory
3. Analyse its current applicability
   - Determine whether rendered images generated using available DEMs and rendering software are representative of their associated real images
   - Determine the navigation accuracy when using real data

1.3. Report Structure

To provide a reference point for the subject of this research, Chapter 2 contains a brief overview of the missions and developments in the context of PPL. The reference missions used to accomplish the above defined goals are described. This information is then used to derive the mission and system requirements.

In Chapter 4.1, the relevant reference systems are defined. Chapter 4.2 provides a description of the camera

model used, including how to derive 3D world coordinates from 2D image coordinates and how to handle distortion. The concept of features, in which the proposed algorithm is based, is introduced in Chapter 3. This chapter provides an overview of the two detectors considered and of the method used to match the detected features between different images. Chapter 4.4 describes the algorithm that uses these matches to find the camera pose associated with a given image.

The data used to analyse the theoretical potential of the software was obtained in the Testbed for Robotic Optical Navigation (TRON) lab. This facility is described in Chapter 5, along with the method used to calibrate the cameras.

Information on the software developed is found in Chapter 6. This includes the software requirements; the architecture and data flow; the acceptance, unit and system tests performed and the validation. A study of the influence of different parameters of the algorithm is made in Chapter 7.1, which culminates in a coarsely tuned set of values, based on a grid search. These values are the ones considered for the remainder of the study. The results of the analysis mentioned as research goals are given in Chapter 7.

Chapter 8 presents the conclusions for this thesis. Recommendations for future studies are given in Chapter **??**, along with some lessons learned. To aid future use and enhancement of the software developed, Appendix A provides a guide of its use in a step by step manner. Appendix B describes how the optimised trajectories were transformed into KUKA commands and divided into sections according to the limitations of the Laser Tracker, as well as the issues that arose in the lab.

2 | Mission Heritage

In this chapter the current research will be placed in the context of the past achievements in planetary landing, the current developments of optical navigation systems and expected future missions in which these advances will play a role. This information is also used in guiding the approach taken for this research, namely in the choice of a reference mission and the definition of the mission and system requirements.

However, given the novelty of the approach and the decision to focus the research on the isolated navigation system (motivated in Section 1.2), rather than integrating it into a GNC system or with other measurements using a filter (as was done in all the examples found), most of the methods used in the research are not based on this information.

Section 2.1 provides an overview of the most relevant landing missions, their landing accuracy and the methods used to achieve it. The different types of algorithms currently being developed for optical-based TRN are characterised and exemplified in Section 2.2. A brief mention of the missions planned for the future in the context of PPL aided by optical-based navigation is given in Section 2.3.

The reference missions are described in Section 2.4 and the requirements for the mission and the system are defined in Section 2.5. Finally, Section 2.6 defines the constraints imposed to the current thesis.

2.1. Previous Missions

The first landing missions had the Moon as the target. On 31 January 1966, Luna 9 performed the first lunar landing [Siddiqi, 2002, p. 53]. With the sole intention to safely land on the Moon, the controls were based on altitude information aquired through a radar system[1]. A few months later, on 2 June 1966, NASA achieved the same with Surveyor 1 [Siddiqi, 2002, p. 55]. The touchdown happened 14 km away from the target. The control was achieved through altimeter and Doppler velocity-sensing radar systems: it was based on altitude and velocity information.

In other words, for these missions, the vehicle's trajectory was planned prior to launch and was not corrected autonomously. The control was limited to initiating descent upon reaching a predetermined altitude. Since there are perturbations on the trajectory, the point where descent is initiated will not be the one planned for. Thus, the touchdown will not correspond to the target point.

The first PPL was achieved as part of the Apollo 12 mission [Bennett and , U.S.]. After analysing the results from the previous mission (Apollo 11), methods to correct navigation were developed. This led to a landing ellipse of about 2.4×6.7 km (1.3×3.6 nautical miles). This example gives a clear indication of the impact of navigation accuracy on the landing precision. However, the mission was manned and used inputs from the crew, even during automatic guidance. An IMU functioned as an autonomous sensor. Its outputs were fused with the manual inputs.

The first application of optical navigation for space exploration was done for the Mars Exploration Rovers (MER) in 2004. The Descent Image Motion Estimation System (DIMES) was developed by Jet Propulsion Laboratory (JPL) to address the challenges posed by the steady-state Martian winds [Cheng et al., 2004]. By measuring the steady-state horizontal velocity, the Transverse Impulse Rocket System (TIRS) could be fired to compensate it. This system was, in fact, fired during the landing of the Spirit spacecraft allowing it to land safely.

The camera chosen was the MER navigation camera, which had a 45° Field of View (FOV). The navigation system also made use of an IMU and a radar altimeter system. The radar provided altitude measurements, and with the IMU the orientation of the lander with respect to the ground was determined by integrating the angle rates.

The captured images are first processed to remove effects such as the higher brightness at the spot immediately below the Sun and the shadow of the lander. Then a high-contrast location is selected in the first of the three images used, shown in Figure 2.1 (left) for the first descent image of MER-A Spirit. The same location is found by correlation with the next image (right image of Figure 2.1). The horizontal displacement between the two images provides the velocity of the lander. This is done separately twice for each consecutive pair of images, resulting in four values that are compared among themselves and with the IMU measurements. If the values are in agreement, the velocity is provided to the TIRS.

[1] /https://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=1966-006A, last access: 18/04/2018

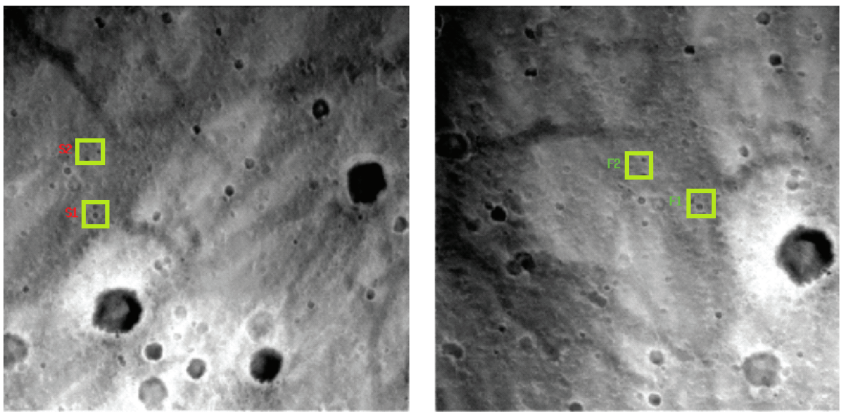


Figure 2.1: Features selected from first image (left) and found in second image (right) from the descent of MER-A Spirit [Cheng et al., 2004]

The images were taken during the parachute descent, starting from around 2 km high and separated by 3.75 s (which corresponded to heights of about 1720 and 1440 m). To allow for a meaningful correlation between different images, the estimated position and attitude of the lander was used to registrate each image (warp it to correspond to the ground plane).

Finally, on 14 December 2013, the Chinese CE-3 lunar lander successfully achieved the first autonomous PPL, having touched down 89 m from the target site [Li et al., 2015]. Since it was an unmanned mission, this accuracy could only be achieved by a fully autonomous GNC and by correcting IMU measurements during the PD and soft landing.

As is noted by [Li et al., 2015], the state of the lander during these last phases changes too fast for communication with the ground station to be viable. In other words, the lander cannot be accurately commanded via Telemetry Track and Command, which justified the need for an autonomous GNC system. The last update from ground was received about 10 minutes before PD, which began at around 15 km of altitude.

This autonomous GNC system was based on an IMU. Starting from around 8 km of altitude, the bias and drift of the IMU began being corrected through laser and microwave ranging sensor measurements, which also allowed for high-precision navigation. From around 4 km of height, the velocity data was improved via a microwave velocimeter sensor, without which the success probability of the mission would have only been 10 %. This last fact is a clear indicator that IMU information by itself is not sufficient for PPL.

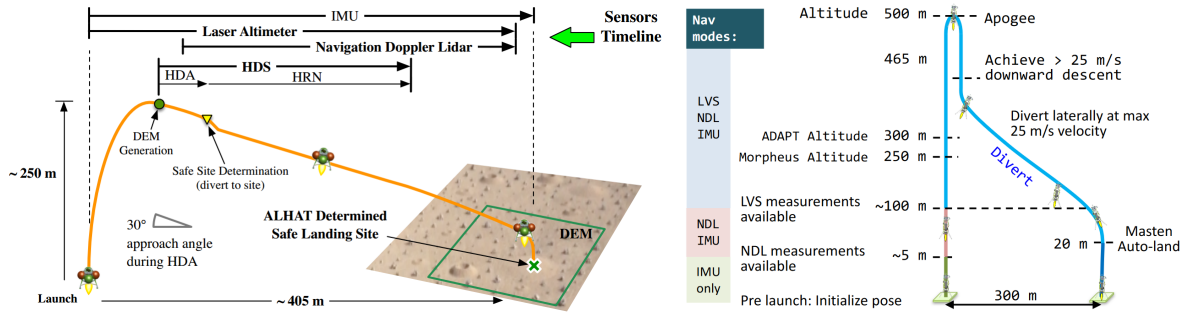
Thus, accurate navigation solutions are valuable to periodically update the state of the lander and achieve PPL.

2.2. Current Developments

As mentioned in Chapter 1, one of the navigation systems currently under development is NASA's ALHAT. With the objective of accomplishing autonomous safe precision landing on solid solar system bodies under any terrain lighting conditions [Carson et al., 2015], the system includes a specialised navigation filter and various sensors. The current suite of sensors includes a lidar-based Hazard Detection System (HDS), a Navigation Doppler Lidar (NDL) velocimeter, a long range Laser Altimeter (LAlt) and a Lander Vision System (LVS) used for TRN, consisting of an optical camera [I. Restrepo et al., 2018]. The system is set to achieve global precision landing – with respect to the intended landing site – of 90 m and local precision of 3 m – with respect to the safe landing site selected by the HDS [Carson et al., 2014b]. Other sensors such as an IMU and a Star Tracker are also included.

The generic mission for which ALHAT is being developed begins TRN at altitudes of around 20 km, via the LVS and LAlt. The NDL provides high accuracy velocity measurements starting at 2-4 km of altitude until 30 m (this lower limit is imposed by the effect of the lander plume on the measurements). The importance of accurate velocity measurements has been made clear by Li et al. [2015] in the context of CE-3's success. The HDS is to operate starting from 1 km ranges to the target site until 100 m. The final portion of the trajectory relies only on IMU measurements [Carson et al., 2014b].

As stated by Carson et al. [2014b], the performance of ALHAT depends on the performance of each of the individual sensors, which is true of any complex system. As such, each individual sensor underwent extensive



(a) Morpheus test [Carson et al., 2015]

(b) COBALT test [I. Restrepo et al., 2018]

Figure 2.2: Flight profile for ALHAT tests

testing and calibration. The ground and flight tests were performed alongside development and allowed for the improvement of the designs and algorithms, as well as the tuning of the parameters involved. Two instances of each sensor were built, both to ensure the existence of a back-up and to speed up the testing and development process. Simulations of the sensors provided additional information during the development.

Apart from the individual performances, their integration into the vehicle is also of major importance. Six tests were performed in 2014, in which the sensors (apart from LVS) were fully integrated into the Morpheus rocket-propelled Vertical Testbed (VTB), three of which were closed-loop tests, including one performed during the night [Carson et al., 2015]. The tests were performed on a terrain representative of a lunar maria, with a descent trajectory starting at around 250 m of altitude and 405 m from the intended landing site. The flight profile and sensor timeline can be seen in Figure 2.2a. For safety reasons, an independent navigation system based on Global Positioning System (GPS), IMU and altimeter measurements was run alongside the ALHAT navigation system. An Autonomous Flight Manager (AFM) was used to switch to this independent system if the safety conditions required it. This same navigation system was used for the open-loop tests.

For the physical integration of the sensors, alignment features were used. These allowed to relate the different refference systems associated to each of the sensors to the vehicle's frame.

The tests were successful in achieving the objectives, namely precision landing through the ALHAT system and final landing near the site selected by HDS.

In 2017, three new tests were performed on the same system, this time integrated into CoOperative Blending of Autonomous Landing Technologies (COBALT) on board the Masten Xodiac suborbital rocket testbed [I. Restrepo et al., 2018]. The sensors tested were NDL and LVS. The tests were all open-loop tests and began at an altitude of 500 m, as seen in the flight profile shown in Figure 2.2b. In this test, the TRN measurements were shown to be essential for PPL, since the position error began to drift at the moment the measurements were no longer available. These measurements were used as filter updates every 1-2 s.

Another system for which closed-loop flight tests have been performed is Autonomous Terrain-based Optical Navigation (ATON) currently being developed by Deutsches Zentrum für Luft- und Raumfahrt (Germany Aerospace Center) (DLR). The navigation system is intended for landings on solar system bodies with no or very thin atmosphere [Theil et al., 2017]. The required landing accuracy is 100 m. This requirement translates to an accuracy of 1% and 0.5% of altitude for cross-range and height, respectively, during Descent Orbit (DO).

Although the system is being developed for general use, a reference lunar mission has been defined, which has been used for the simulations and tests performed. The landing mission begins in a 100×100 km parking orbit, followed by a 100×10 km DO. The PD begins close to the pericenter and, for the final 100 s, the lander is almost vertically aligned with the surface. At an altitude of 1 m, the required velocity is of 1 m/s.

To achieve the required landing accuracy, the system shall include an IMU, a star tracker, a laser altimeter, a flash Light Detection and Ranging (LIDAR) and a monocular camera. The camera has a resolution of 1024 × 1024 px, a FOV of 40° and a frame rate of 30 Hz. From the images captured by this camera, the latest available after the computation of a navigation solution is used for TRN, resulting in an effective frame rate between 3 and 5 Hz. The algorithm, Crater Navigation (CNav), is based on the detection of light and shadow areas of impact craters.

These areas are isolated and associated with a centroid. Light and shadow centroids are paired based on their distance; and the pairs are used to define a vector from the shadow centroid to the light one. The illumination direction of the image is defined as the maximum in the direction histogram of these vectors. The pairs

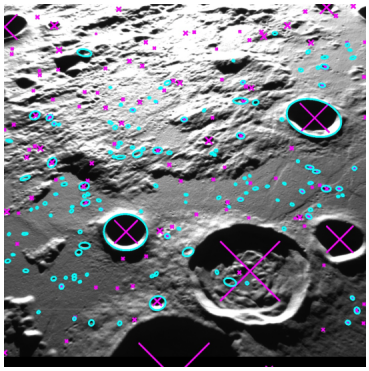


Figure 2.3: Detected (blue ellipses) and database (pink crosses) craters during simulation in TRON [Theil et al., 2017]

Figure 2.4: Example of "crater" used for ATON flight test [Trigo et al., 2018]

for which the direction is within a certain range of the illumination direction are used to fit ellipses around the identified areas. These ellipses (expected to be craters) are matched to a database, generated offline, containing the crater parameters (position and radius). From the matches, the camera position and attitude, and thus the lander's, can be determined using Effective Perspective n-Point (EPnP) along with the Quaternion-based Characteristic Polynomial (QCP) solver [Trigo et al., 2018].

The testing efforts begun in a simulation environment, which included the dynamical model of a lunar landing vehicle and of its navigation sensors. The artificial images were rendered based on DEMs from the Kaguya and Lunar Reconnaissance Orbiter (LRO) missions, the camera parameters and the vehicle's state. Since at low altitudes, the Kaguya DEMs are too noisy, they were enhanced with an artificial structure at the landing site (details of which are given by Lingenauber et al. [2013]). The LIDAR and laser altimeter measurements were obtained from the depth data generated alongside the artificial images.

The second step was to embed the image processing and navigation into a Matlab/Simulink simulation environment. First this was limited to open-loop simulations using pre-rendered artificial images. Then, the loop was closed and image rendering was included. Due to this last addition, several days were required to simulate a full 600 s PD.

Next, the system was tested in TRON (see Chapter 5), for which the artificial images in the previous test were replaced by real images captured in the laboratory. Figure 2.3 shows an image obtained during this simulation. The blue ellipses represent the craters detected during the flight, whereas the pink crosses mark the database craters. Matches were found for the overlapping symbols. The trajectory was divided into three separate parts, each tested with different terrain models and different scales. Finally, the system was tested in six flight tests on-board of an unmanned SwissDrones (SDO 50 V2) helicopter [Theil et al., 2017]. The objective was to demonstrate the real-time close-loop operation of the system.

Eighty "craters" (Figure 2.4) with diameters from 0.5 to 5 m were randomly scattered throghout the terrain. For the test, the crater database was constructed based on their centres' coordinates, measured using a Global Navigation Satellite System (GNSS) receiver, using a Satelliten Positionierungsdienst der deutschen Landesvermessung (SAPOS) correction signal, and a tachymeter. The position accuracy achieved was 0.01-0.02 m. The ground truth was obtained via the same GNSS receiver, later fused with the IMU measurements. The trajectory stretched through 200 m, starting from 50 m of altitude until around 10 m, from which the helicopter descended vertically down to 1 m. Some biases were found in the state estimate errors, which were attributed to mis-calibration of the camera and misalignments between it and the IMU. Overall, the tests were

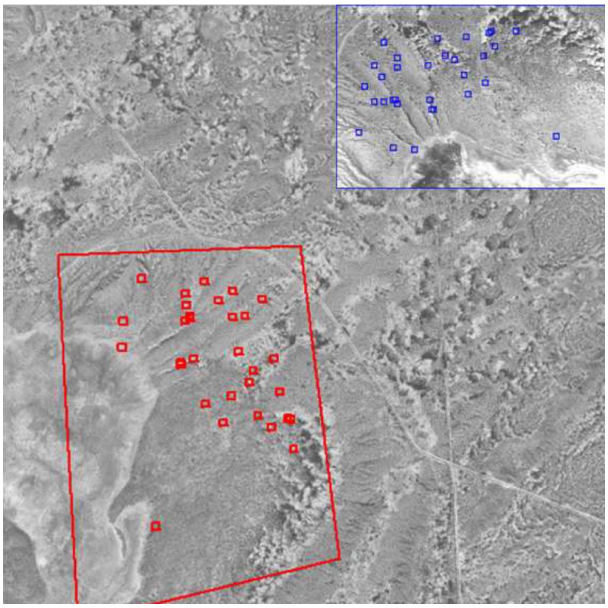


Figure 2.5: Image obtaines at 1600 m during VISINAV test (top right) and matches with the map [Mourikis et al., 2009]

successful.

Another system that was successfully tested in an open-loop flight test, was the Vision-aided Inertial Navigation (VISINAV) system developed by Mourikis et al. [2009]. The navigation algorithm uses camera images to update IMU measurements through an EKF. Two methods are used to derive the lander's state from the descent images: Mapped Landmarks (ML) are features stored in a database constructed offline for which the 3D coordinates are known; Opportunistic Features (OF) are features found in consecutive descent images.

There are also two modes of operation for the ML based navigation: one for nominal operation and another for lost-in-space. For the second case, an initial coarse estimation of the lander's state is obtained via a Fast Fourier Transform (FFT) map matching to reduce the search space of the nominal algorithm [Mourikis et al., 2009]. The map matching is performed using the Harris operator, image geo-registration and spacial correlation. Whereas MLs are used to calculate the absolute position and orientation of the lander, OFs are used to track the relative motion of the lander.

The test performed in April 2006 used a GPS to obtain ground-truth position data. The camera, pointed towards nadir, captured images at 30 Hz with a resolution of 768×484 pixels, corresponding to a FOV of $38° \times 24°$. The IMU had a rate of 50 Hz and the GPS 10 Hz. A GPS time tag was used to synchronise the measurements, which were downlinked in real-time and processed offline.

The experiment was performed at the White Sands Missile Range in New Mexico, onboard a Terrier Orion Sounding Rocket, which flew to a height of 123 km. During the initial portion of the flight, only IMU measurements were taken. Integrating these measurements, the resulting navigation solution was very accurate in comparison to the GPS measurements until the deployment of the main parachute at 4.2 km. From this point, the position error increased significantly (error of about 2700 m).

At 3.8 km of altitude, the ML VISINAV navigation began, tracking 40 MLs at 3 Hz. Figure 2.5 shows a camera obtained at about 1600 m (top right) and the matches made to the map. The blue squares are the features found in the camera image, whereas the red ones are the corresponding features in the map. The red rectangle in the map represents the camera view.

After 5 s, the navigation solution converged from the error in the order of km to about 18 m from the GPS measurements. At 3.1 km of altitude, the VISINAV algorithm is stopped and isn't used until an altitude of 1.6 km. This period of inactivity was included to test the ability of the algorithm to redetect MLs. An equivalent example in a mission scenario is the temporary loss of LOS during major course corrections.

From this height, the ML navigation is resumed and continues until 230 m. At 330 m, the OF navigation begins. Although the accuracy decreases with the number of MLs detected, the 3σ error is reduced from around 550 m (in position) and 10 m/s (in velocity) at the end of the intermediate IMU-only period to 4 m and 0.25 m/s, respectively. The final errors at touchdown were 6.4 m in position and 0.16 m/s in velocity, in comparison

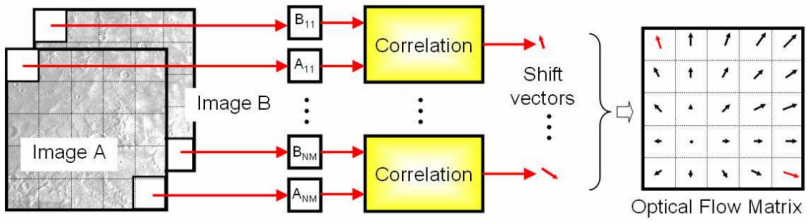


Figure 2.6: Optical flow method used by Tchernykh et al. [2006]

with over 9 km and 30 m/s, respectively, for the IMU-only navigation solution.

Although no attitude ground-truth is available, the estimated associated accuracy bounds are of ±0.15° for each axis. During the intermediate period, the bounds were at ±0.9°.

Apart from the flight test, the FFT algorithm was tested for sensitivity to planetary terrain appearance, illumination and altitude errors. In terms of terrain appearance, the first test was made by determining the percentage of matches in a Monte Carlo run with 850 images between orbital images and images rendered in the simulator introduced by Willson et al. [2005], which aided in the development of the DIMES optical navigation algorithm. The simulator, MOC2DIMES, uses the camera configuration, a descent trajectory and a terrain image (captured by a narrow angle camera on-board the Mars Global Surveyor) to produce three images, among other outputs useful for comparison with the DIMES outputs. Matching rates were above 87%.

The second test was made by matching two images of the same terrain taken at two different time epochs. The test is intended to show the insensitivity of the FFT algorithm to changes in the terrain appearance over time. An example was the changes caused by dust devil tracks for Mars, shown by a successful match between an orbital image taken in 2002 and a descent image taken in 2004 by DIMES. On the other hand, the flight test itself used a map constructed 5 years prior, showing that the effects of human activity, weather and vegetation growth did not have a significant impact on the navigation solution.

The algorithm showed insensitivity to illumination changes even at 180° changes in the incidence angle, which is achieved using local image normalisation. To test for the effect of altitude inaccuracy, the rendered descent image was scaled to a factor from 80% to 120% of the correct one and Gaussian noise was added. The solutions were acceptable for scale errors, and thus altitude errors, of ±5%.

As is evident by these and other examples, optical navigation plays a major role in the current developments of navigation systems for planetary landing. Optical navigation can be achieved through both active or passive sensors. An example of an active optical sensor is the LIDAR used, for example, to create DEMs. Cameras are passive sensors used to capture images.

The information captured by either type of sensor can be processed for optical navigation in different ways. Depending on the method used, current developments can be placed into one of three categories: terrain matching, feature tracking and crater matching.

Terrain matching consists of finding the correspondence between the terrain visible and a map to determine the lander's state with respect to the target. This can be done either with 3D data (in the form of DEMs and 3D models) or with 2D images. An example of such method is proposed by [Tchernykh et al., 2006].

This algorithm begins by dividing two consecutive images into various fragments, as shown in Figure 2.6. Each one of them is correlated between images to find the shift vectors (representative of the changes observed between the images). A matrix containing these vectors (the optical flow matrix) is then used to reconstruct a 3D model of the observed surface.

The estimated state of the lander is used to transform this model from the camera frame to the target frame. The differences between this transformed model and the reference one (map) are then used to improve the estimated state.

Another example of this type of algorithm is NASA's Autonomous Precision Landing Navigation (APLNav), being studied as part of the ALHAT project [White et al., 2009]. In this case, two images are simultaneously captured by two different cameras. The relative positioning of these cameras is shown in Figure 2.7. The estimated state of the lander is used to render the corresponding images at that time. These rendered images are then correlated to the real ones to correct the estimated state. This is an iterative process that culminates in a navigation solution, namely the position and angular state of the lander.

An example of feature tracking is DIMES [Cheng et al., 2004]. The already flown algorithm found two pairs of features between three images. The displacements between the matched features were used to determine

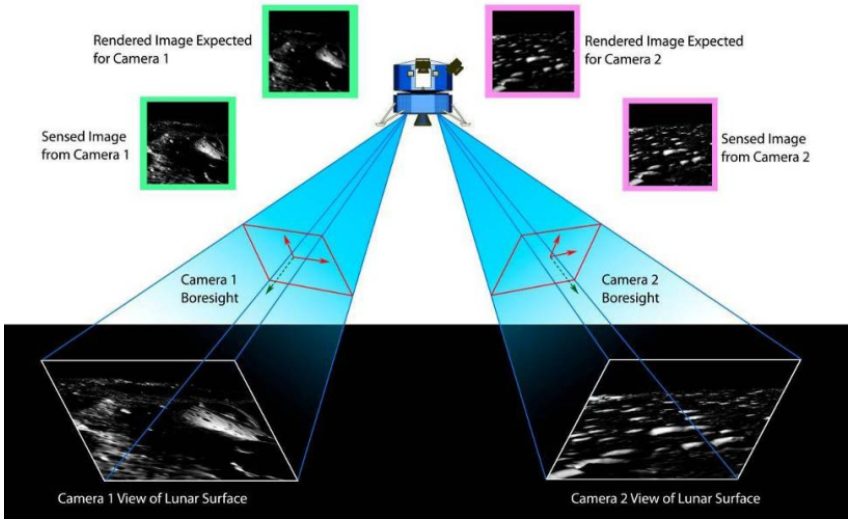


Figure 2.7: Camera positioning in APLNAV system [White et al., 2009]

the velocity of the lander. ESA's Navigation for Planetary Approach and Landing (NPAL) project is a study of a similar approach [Astrium, 2006]. In this case, up to 50 features can be used. These are detected using a Harris corner detector.

Finally, crater matching consists in detecting and characterising craters during the landing and then matching them to a database. An example of such an approach is introduced by Cheng and Ansar [2005]. From each image taken by the onboard camera, a crater detection algorithm is run. This algorithm fits ellipses in the recognized craters and the information is expressed through their centre, radius and orientation. Later they are matched with a crater database. The matches are then used to determine the navigation solution. The algorithm currently being developed by DLR is one of the few examples that use crater navigation without requiring an initial state estimate.

A fourth method for which no example was found is feature matching. This approach uses features (as with feature tracking) and matches them to a database (as with crater matching). The lack of heritage for this approach motivated choosing it for this research.

2.3. Future Missions

As already mentioned, both NASA and ESA are making efforts towards implementing and flying TRN solutions to the PPL problem.

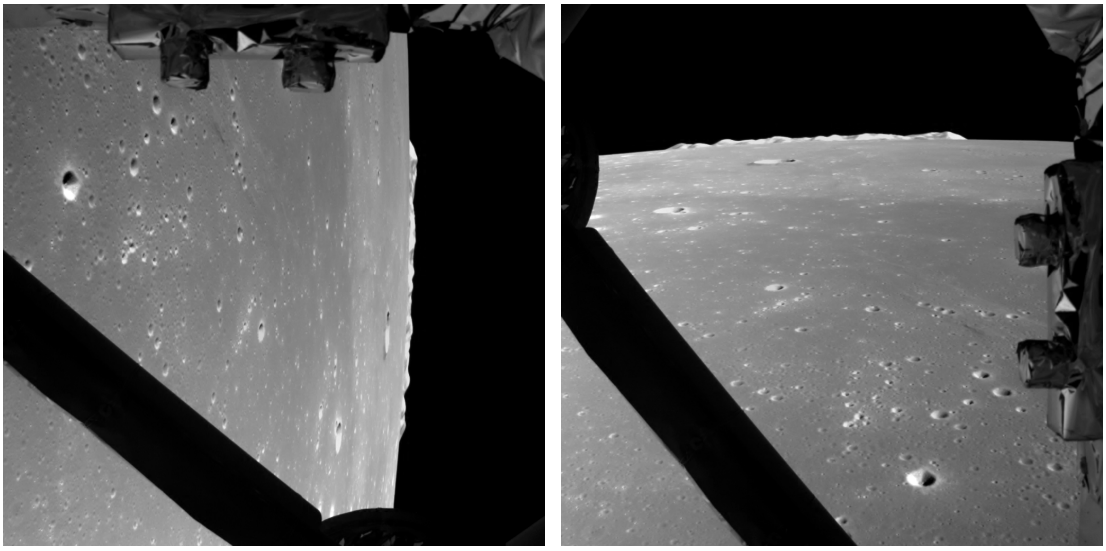
NASA's ALHAT project seeks to achieve a landing precision of 30 m [Epp and Smith, 2007], regardless of surface and illumination conditions. This system includes TRN and HDA and will be used for the Mars 2020 mission. On the other hand, ESA's PILOT is to be incorporated into the future Russian lunar landers.

Small Lander for Investigating Moon (SLIM) is a future mission to be pursued by Institute of Space and Astronautical Science (ISAS)/Japan Aerospace eXploration Agency (JAXA) [Sakai. et al., 2015]. The goal is to demonstrate PPL on the Moon. The TRN will be based on crater matching and a new radar system is to be used. Once again, the lander will perform both TRN and HDA.

2.4. Reference Mission

Chang'e-3, for which some relevant data is available, is the most recent successful lunar landing. Having achieved PPL, the mission is a good example of the ones for which the application is intended. It can be used as a reference mission. Doing so will allow for the analysis of the algorithm when using the type of data available at the date of its development; thus, showing the effect of lower quality DEMs, approximate nominal trajectory data and difference in the real surface texture and the modelled one.

During the landing, 4672 images were taken by the Descent Camera (Descam). These images are available for download at `/http://moon.bao.ac.cn` and will be used to analyse the current applicability of the naviga-



(a) Downloaded image (b) Processed image

Figure 2.8: Image number 300 taken by the Descam on board of Chang'e-3. [2]

Table 2.1: Descam camera parameters [Liu et al., 2015]

Items	Value	Distortion coefficients	Values
Resolution (pixels)	1024×1024	k_1	3.00407×10^{-3}
Pixel size (mm)	0.0067	k_2	1.38071×10^{-5}
Focal length (mm)	8.5952	k_3	1.63601×10^{-7}
Principal point offset in u (mm)	-0.0130	p_1	-4.11569×10^{-5}
Principal point offset in v (mm)	-0.0422	p_2	-2.52348×10^{-5}

tion software (as proposed in Chapter 1). It is to note that these images are mirrored. Before being used they were rotated 90° counterclockwise and flipped horizontaly. This image orientation was also used by Liu et al. [2014]. An example of an image processed in this way is shown in Figure 2.8.

Liu et al. [2014] provides ground-truths for the lander's trajectory. The available plots are shown in Figure 2.9. The error bars are magnified 10 and 1000 times in Figures 2.9a and 2.9b, respectively. For each measurement the position of the vehicle is given in terms of its spherical coordinates (latitude, longitude and altitude). The longitude changes very little and is about 340.5°; the latitude ranges from about 42° to 44°; and the height from about 12.5 km to 0. The red circle corresponds to the hovering stage of the landing.

It is important to note that the altitude is measured relative to the lunar surface and not to a reference sphere or ellipsoid. This is made evident by Liu et al. [2014] when discussing the altitude measurement differences between the reconstructed trajectory and the altimeter data (which resulted from the slight inclination of the vehicle with respect to the normal to the surface).

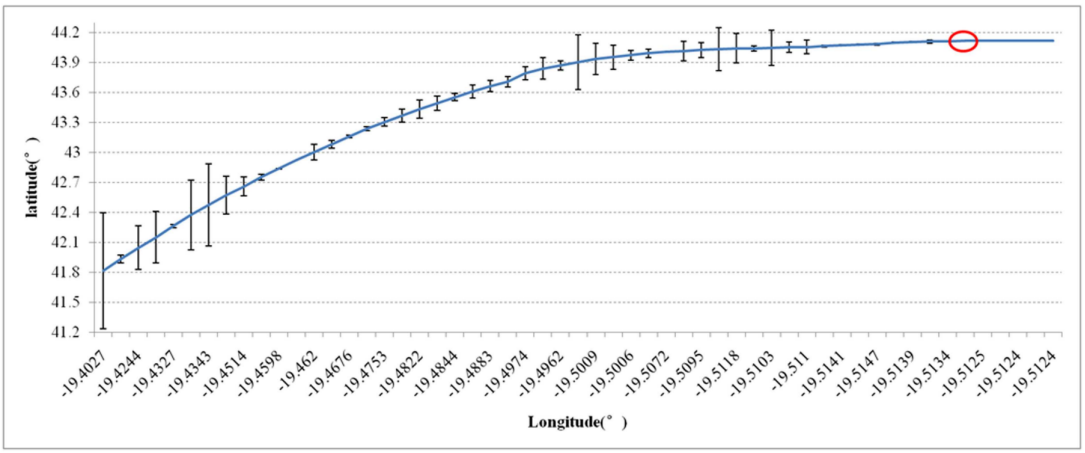
The camera parameters of the Descam are given by Liu et al. [2015], of which the significant ones are summarised in Table 2.1. The camera has a resolution of 1024 × 1024 px. From this value, the pixel size and the focal length, the FOV is of around 45°. The principal point offset values are given with respect to the centre of the image. The distortion parameters are given in the same format used in Open Source Computer Vision Library (OpenCV). The frame rate was 10 fps. The camera was placed on the bottom of the lander, aligned verticaly.

2.5. Mission and System Requirements

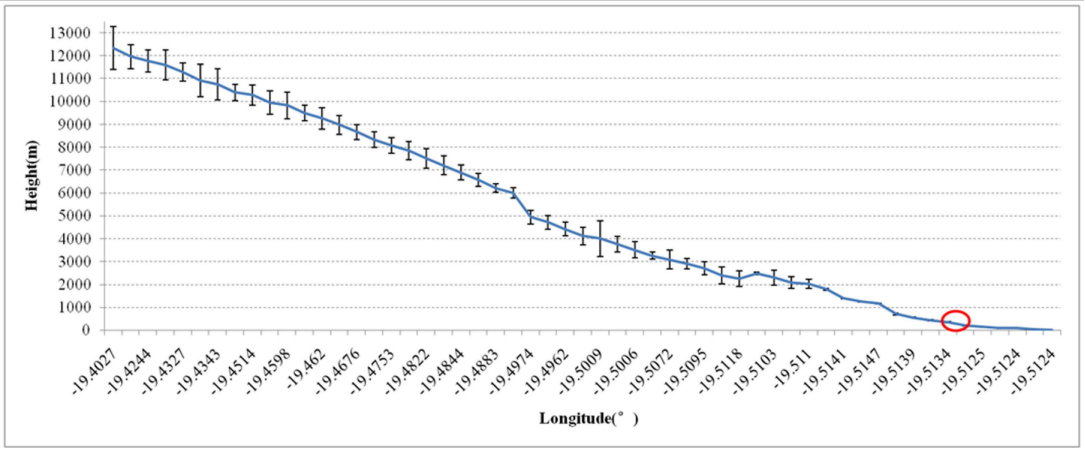
From the research question defined in Chapter 1 the following **mission requirements** (MR) and **system requirements** (SR) were derived:

- **MR_1** – The navigation system shall perform TRN

[2] /http://moon.bao.ac.cn/cedownload/LCAM/2B/CE3_BMYK_LCAM-300_SCI_N_20131214130537_20131214130537_0001_A.2B, last access: 18/04/2018



(a) Latitude as a function of longitude (uncertainty bars are magnified 10 times)



(b) Height as a function of longitude (uncertainty bars are magnified 1000 times)

Figure 2.9: Ground-truth available for CE-3 landing trajectory. [Liu et al., 2014]

  - **SR_1a** – The system will only use the Moon-Centred Moon-Fixed (MCMF), camera, vehicle specific and image related (2D) reference frames
  - **SR_1b** – No knowledge of the gravity field shall be required
  - **SR_1c** – No knowledge of the vehicle's dynamics shall be required
  - **SR_1d** – No ephemerides data shall be required online
  - **SR_1e** – The navigation system shall be based on camera images with 1024 × 1024 resolution
- **MR_2** – The navigation solution shall not require an initial state estimate
  - **SR_2a** – No initial estimates of the position and/or orientation shall be required
  - **SR_2b** – Individual navigation solutions shall be uncoupled
- **MR_3** – The navigation solution shall be adequate to fuse with IMU measurements for PPL purposes
  - **SR_3a** – The solution shall have an average error below 1% of the LOS distance to the target surface
  - **SR_3b** – The solution shall have a maximum error below 3% of the LOS distance to the target surface apart from clear outliers
  - **SR_3c** – The average time per pose required to derive a solution shall be below 10 s on DLR's optical navigation demonstrator
- **MR_4** – The navigation system shall be used for lunar landings
  - **SR_4a** – The navigation system shall yield a solution starting from a 100 × 100 km parking orbit
  - **SR_4b** – The navigation system shall be robust to the expected state deviations from the nominal trajectory at the beginning of the PD
  - **SR_4c** – The navigation system shall be robust to the expected state deviations from the nominal trajectory at the beginning of vertical descent
  - **SR_4d** – While in parking orbit, an initial estimate of the position of the lander with an error of 50 km can be used to select the appropriate region-specific database from a super-database (covering the full lunar surface)
- **MR_5** – The navigation system shall be used for asteroid landings
  - **SR_5a** – The navigation system shall be robust to the expected illumination changes

To perform TRN, the essential source of information must be the terrain of the target body. The physical interactions between it and the lander – manifested by the gravity field and the vehicle's dynamics (**SR_1b** and **SR_1c**) – or the location of these bodies relative to others in the Solar System – available in the form of ephemerides data (**SR_1d**) – are irrelevant. As such, the only reference systems that should be involved in the pose determination are the ones relating to the target body (fixed with respect to its surface), to the camera (including the associated 2D image frames) and to the vehicle (**SR_1a**).

It was noted that the current methods for TRN in the context of planetary landing rely on initial state estimates. In other words, these estimates are active inputs in the pose determination. This characteristic makes the system unusable for the *lost in space* scenario. One of the objectives of this research (stated in the research question in Section 1.2) is to determine the accuracy that can be achieved without using initial state estimates (**SR_2a**). Since they are independent from the estimated state of the lander, the raw navigation solutions (before being fused with other measurements in a filter) should be uncoupled from one another (**SR_2b**).

This is not incompatible with requirement **SR_4d**, since the suggested approach here provided is used only to choose between the databases used, rather than as a direct input to the pose determination. The accuracy required for the state estimation is very low compared to algorithms such as the ones used in NASA's ALHAT (see Section 1.1). On one hand, with a super-database of the lunar surface, large deviations of the lander from the planned trajectory upon insertion into the parking orbit will not invalidate the use of the navigation method. On the other hand, fragmenting this super-database into region-specific ones would result in major gains in computation time.

Requirements **SR_4b**, **SR_4c** and **SR_5a** address the relevant trajectory and condition deviations at different phases of the landing. A system robust to these deviations improves the probability of mission success.

If the proposed navigation method is included in a landing mission, its outputs would be fused with measurements from different instruments, the most prominent of which is the IMU (used in all examples discussed in previous sections). A good filter can significantly improve the navigation solution. However, the integrated navigation system can only have high performance if the individual elements are accurate themselves. Thus, accuracy requirements can be set for the raw navigation solution. The values specified for requirements **SR_3a** and **SR_3b** were based on the performance found for CNav during the prior internship study.

On the other hand, for real-time use, there must be a requirement on the computation time. Since the navigation method is being developed within DLR, requirement **SR_3c** was obtained after consultation. To test it in a more practical manner, it is assumed that the running time in the mentioned instrument is 10 times

as high as in a desktop computer. Thus, a solution shall be derived in 1 s in this environment.

Considering, once again, the context in which the thesis is being developed, the lunar landing scenario was based on the ATON reference mission (**SR_1e** and **SR_4a**).

Since it is based on camera images, this solution is limited to landing on illuminated surfaces. For the case in study, the images used to generate the database were rendered from a 3D model. This model is based on DEMs of the target surface. As such, the solution is only applicable to surfaces with significant height variations, such as terrain with craters and mountains. However, in principle it would still apply in the presence of unambiguous texture (meaning, different points can be uniquely differentiated) which remains constant and in sight (*e.g.*, not covered by clouds).

2.6. Research Constraints

Given the time limitations of this research and the currently available data and facilities, the following constraints were imposed:

- The performance during soft-landing on a cratered body will be studied based on TRON's terrain model 3 (Chapter 5)
- The performance during PD for the CE-3 trajectory and landing site will be studied based on the available real data
- The robustness of the navigation solution to deviations of the lander's initial state will be analysed based on trajectories generated through a pre-existing optimiser
- The robustness of the navigation solution to changes in the illumination conditions will not be analysed
- The selection of the appropriate region-specific database from a super-database based on an initial guess of the lander's state (for a lunar parking orbit) will not be addressed
- The navigation software will not be integrated into a GNC system
- No close-loop landing simulation will be performed

Despite these limitations, the methods, software and work-flow used in this research can be used in future studies. Appendix A provides a guide to their use.

2.7. Pre-existing Software

Taking into account the concept outlined in Section 1.2, the requirements specified in Section 2.5 and the constraints set on the research in Section 2.6, the choice of some software elements required for the work-flow will be motivated in this section. These includes an image rendering software (to obtain the data images for the database), the feature detectors, the feature matching algorithm, the algorithm for pose determination, the camera calibration tools and image processing software.

Image Rendering Software

To render the data images required to generate the database, a rendering software where DEMs can be used to create a 3D model, and which allows for the specification of camera parameters is required. An open source software that fulfills these requirements is Blender. Apart from providing an intuitive 3D interactive interface, which allows the visualisation of the model and easy inspection of the relative position and orientation of the various elements, such as the camera with respect to the model, it includes a python Application Programming Interface (API). This can be used in real-time through the internal python console or through external scripts. Using external scripts with Blender in the background (meaning that the visual environment is not presented to the user) increases the speed of the operations, especially for image rendering.

Blender can also be used to render depth data of the 3D model, which is necessary to determine 3D world coordinates from the respective 2D image coordinates (see Section 4.3). Finally, it provides a wide range of options in manipulating the lighting conditions and the materials of the objects, including the use of images as textures.

Another rendering tool considered was SensorDTM, a tool developed by DLR. This tool is very limited in options and does not provide a visual representation of the 3D model. However, it was developed for the specific use of planetary DEMs, making it very precise and simple to position the model and the Sun as the lighting source.

Feature Detectors and Descriptors

During the review of the available feature detectors and descriptors, a trade-off between performance, speed and availability was made. The performance parameters considered were based on the transformations expected to be visible in the context of landing images. Thus, the algorithms were chosen based on their invariance to translation, rotation, zoom, illumination changes and affine transformations (perspective changes).

The chosen detectors were Accelerated-KAZE (AKAZE) and Binary Robust Invariant Scalable Keypoints (BRISK), both of which are freely available, have the required performance and speed for real-time use.

Feature Matching

To match a feature found in an online image with the most similar feature included in the database (found in one of the data images), a linear method (searching every possible combination) is not appropriate, given the number of features involved and the intention to use the application in real-time. As such, a different method that reduces the search time without great loss of performance is required. A commonly used tool in computer vision for feature matching is Fast Library for Approximate Nearest Neighbors (FLANN). This library includes different algorithms to organise database features into indeces, to search these indeces for the closest match to a given query point and to optimise the indeces for the search time.

Pose Determination Algorithms

The algorithm chosen to determine the camera pose from the matched features in the proposed application was EPnP. This algorithm has been successfully used as part of the ATON navigation system [Trigo et al., 2018]. It was found to perform well and is sufficiently fast for real-time usage.

Camera Calibration Tools

To determine the camera parameters and the frame transformation between the cameras used in the laboratory and the measurement instrument, an adequate camera calibration tool is required. Given its availability and compatibility with the calibration targets available, the tools to be used are the DLR-internal tools, Camera Calibration Detection Tool (CalDe) and Camera Calibration Lab (CalLab).

OpenCV

OpenCV is, as the name indicates, an open source library that provides resources for computer vision. It includes various feature detectors and descriptors (including AKAZE and BRISK), FLANN and EPnP. Furthermore, the distortion parameters resulting from the calibration with CalLab are given in the format used in OpenCV. Thus, using this library simplifies the development of the proposed aplication, while keeping it from being restricted by licenses.

Image Processing Software

For developing test images and other occasional image processing, the software *irfanView* was used. Apart from its availability, this choice was motivated by the wide range of image processing functions available, including median filters, contrast and brightness adjustments, and a precise cropping tool. It also alows for batch processing and renaming, thus increasing the speed of operations and lowering the possibility for errors.

3 Feature Detection and Matching

To answer the research question, posed in Section 1.2, and fulfill the requirements, defined in Section 2.5, relevant information must be obtained by processing the images rendered (data images) and the ones captured during the landing (online images).

The proposed software is based on the comparison between features found in images captured during the landing and the features stored in a database, which were detected on images rendered during the design phase of the mission, and expected to be representative of the landing ones. Since the software is autonomous, it has to rely on computer vision.

The average human has the capacity to easily identify objects in an image. This ability is often taken for granted and may lead to erroneous expectations of what a computer is able to achieve. Being able to make some very accurate recognitions (Figure 3.1a), the state of the art for recognition Artificial Intelligence (AI) still makes very bad identifications [Girshick et al., 2014]. Figure 3.1b shows an example where two fish are recognised with about the same certainty (76% and 74%) as a goldfish and a dog, respectively. The same AI can even be tricked into identifying noise patterns as objects with over 99.6% certainty (Figure 3.1c) [Nguyen et al., 2015].

Fortunately, the application in study does not benefit from object identification. First, this method does not yield precise positions of the identified objects (which would be necessary to derive a navigation solution). Second, there aren't many types of objects that can be identified in an image of the lunar surface. Third, for pose determination, it is irrelevant whether an image point corresponds to a crater, a mountain or other physical structure.

Instead it should rely on some sort of pattern recognition, in the same way as humans use maps and landmarks to orient themselves in the environment they perceive. But whereas humans can make use of abstractions to represent reality (*e.g.*, lines to represent roads), the same cannot be expected from computers.

For a computer, an image is a collection of pixels and their relative relevance is not obvious. Figure 3.2 (left) shows three points in a simple image that provide different amounts of information. Point A provides virtually no information. While point B contains more, there are still many other points along the side of the square that are identical. If taken by themselves, these two points could lead to the incorrect assumption that the right image in Figure 3.2 was a rotated version of the left one. Point C, although not nearly sufficient, is highly identifiable.

These highly recognisable points in an image can be referred to as image features. They are points that, in some way, contain a high amount of information and that can potentially be identified in a new different image. Depending on the measure used to quantify the information contained in an image point, different points are detected. Among other aspects, these different measures lead to different feature detection algorithms.

From the existing algorithms, two were chosen based on their performance, speed and availability, namely

(a) Correct recognition [Girshick et al., 2014]

(b) Incorrect recognition [Girshick et al., 2014]

(c) Recognition of noise as objects [Nguyen et al., 2015]

Figure 3.1: Examples of image recognition results

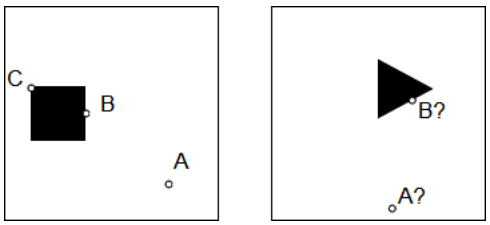


Figure 3.2: Example of points in an image with different amounts of information

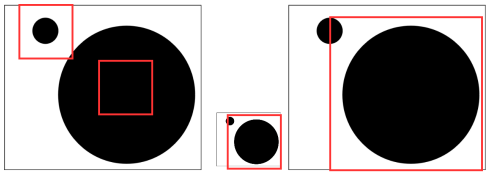


Figure 3.3: Example of features of different scales and different methods to varry the scale of detected features

AKAZE and BRISK. The performance parameters considered were invariance to translation, rotation, zoom, illumination changes and affine transformations (perspective changes), which were the transformations identified as relevant for the application. An algorithm is considered invariant to a transformation if it is sucessful in matching two images, where one is the transformed version of the other.

Speeded-Up Robust Features (SURF) was rejected due to requiring a license for its use. Since both AKAZE and BRISK are freely available in the same library (OpenCV) and the same functions and variables are used for both, working with both does not increase the complexity of the software. This also allows to observe the effect of using different detectors in the final navigation solution.

Section 3.1 provides an introduction to the concept of scale-space used in feature detection. A qualitative explanation of the chosen algortihms is given in Sections 3.2 and 3.3 (AKAZE and BRISK, respectively). The use of masks to restrict the image area for feature detection is explained in Section 3.4. The algorithm used to limit the number of features to be used for the database and feature matching is described in Section 3.5. Finally, the method used to match features between different images is explained in Section 3.6 and the match down-selection algorithm is described in Section 3.7.

3.1. Scale Space

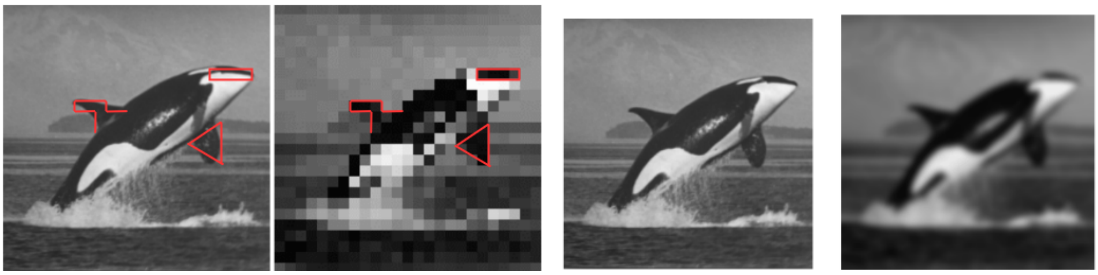
As previously mentioned, for a computer, an image is a collection of pixels. Each pixel by itself normally does not provide any meaningful information that can be used for later matching. Therefore, the information contained in a pixel is determined in the context of the pixels surrounding it.

The area that is considered, however, is still fixed. The consequence of this is shown in Figure 3.3. The original image (left) is composed of two circles of different sizes. Even though both circles could be used as features, only the smaller one would be detected when using the detection area shown by the red squares. To detect the bigger circle, either the image size should be reduced (Figure 3.3 middle) or the detection area (filter size) increased (right).

Increasing the filter size is straightfoward and intuitive. Despite that, certain algorithms used for detection do not allow re-scalling the filter without changing its properties (as a consequence of their discrete nature). A consequence of this would be scale-variance. For example, if we zoomed in an image by 200%, in theory we should detect the same features using a 5 ×5 filter in the original image and a 10 ×10 one in the zoomed image.

The first problem arises from the fact that a 10 ×10 px filter is not feasible: the side needs to have an odd number of pixels, so that a central pixel exists (which corresponds to the possible feature point being evaluated). Second, even if such a scalling was possible, since the properties are changed, a point that satisfies the conditions being evaluated in one scale, could not do so in another. Third, even if it did, the descriptor used to characterise the feature for later matching could be very different in both cases.

This inability to scale the detection area used is observed in both AKAZE and BRISK. The alternative, again, is to decrease the image size or, more generally, the resolution. Most commonly, resolution is referred as a measure of the number of pixels in an image. In general, it can be thought as the amount of information available in it. Thus, one way to reduce the resolution is by down-sampling the original image. An example of



(a) Down-sampling (b) Gaussian blur

Figure 3.4: Two methods of reducing image resolution[1]

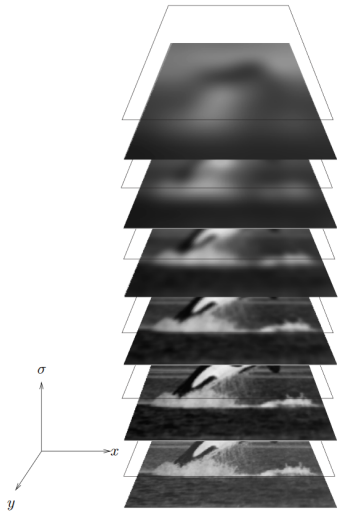


Figure 3.5: Example of a scale-space[1]

this is shown in Figure 3.4a.

As is made clear through the red lines, this method can significantly change the structures represented in the image (spurious resolution). As a consequence, it is not a suitable method for feature detection and matching between images. An alternative (which benefits from the second definition of resolution) is Gaussian convolution, which blurs the image, thus reducing the information stored in it. Despite keeping the total number of pixels constant, the smaller details of the image disappear as a consequence of the blurring. In this way, the image resolution is lowered. An example of this method is shown in Figure 3.4b. Unlike in Figure 3.4a, the structures of the original image are not altered. Once the image has been blurred, it is then possible to down-sample it without causing spurious resolution.

Regardless of the method used to detect larger structures as features, the smaller ones become too small and are no longer recognised. For features of multiple scales to be detected, a scale-space needs to be built, an example of which is shown in Figure 3.5. A scale-space can be seen as a 3D representation of the image, where the third dimension represents the resolution of the image. The new variable, scale σ, increases with decreasing resolution, starting at 0 for the original image, and can be used to describe the scale at which a feature is found.

The scale-space is built by sequentially reducing the resolution of the previous image and layering the results. Despite being built in a discrete way, information can be interpolated along the scale dimension (as well as along the spacial dimensions), resulting in sub-scale (and sub-pixel) accuracy.

3.2. AKAZE

AKAZE was developed to increase the speed of the KAZE detector, which in turn was designed to increase the distinctiveness and repeatability of the detections found by methods such as Scale-Invariant Feature Trans-

[1] /http://www.cs.uu.nl/docs/vakken/ibv/reader/chapter9.pdf, last access: 18/04/2018

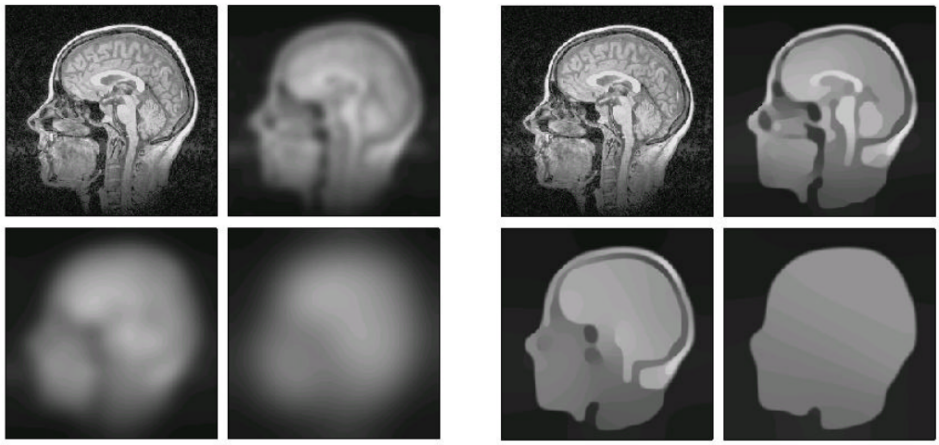


Figure 3.6: Comparison between linear and non-linear diffusion for different scales. [Alcantarilla et al., 2013].

form (SIFT) and SURF. The increase in performance observed in KAZE was achieved by using nonlinear diffusion instead of the regular Gaussian blurring to construct the scale-space. An example of the difference between the two methods is shown in Figure 3.6. The method used to solve the nonlinear diffusion equation in KAZE, however, was very computationally demanding, making it a very slow algorithm. A new solving method was developed, thus leading to the development of AKAZE. This new method increases the speed by several orders of magnitude without major changes in performance.

The concept used to reduce the image resolution is explained in Subsection 3.2.1, followed by a brief discussion on the construction of the scale-space in Subsection 3.2.2. Finally, Subsections 3.2.3 and 3.2.4 provide an overview of the methods used to detect and describe the features.

3.2.1. Nonlinear Anisotropic Diffusion

Diffusion is the process whereby some variable which varries along a given space is redistributed into a more uniform configuration. Examples are the transfer of heat from hot to cold objects or the movement of salt mollecules from high to lower concentration zones. In the context of image processing, diffusion corresponds to making the pixel intensity distribution more uniform. Using a Gaussian filter to blur an image has this effect. Doing it, reduces the amount of small details (variations in pixel intensity within a small area).

In nonlinear diffusion, this process is faster depending on the position of a point: the speed of heat transfer for an object made of different materials can be different at points where the conductivity is different. An example for the image case would be varrying the size of the Gaussian filter depending on the position of the pixel in the image. A smaller filter is equivalent to a slower mixing, resulting in less detail reduction.

Anisotropic processes are those that have different strengths according to the direction they are occurring. Wood is an anisotropic material, since it has higher resistance in the direction along its fibers in comparison to the one perpendicular to them. To derive the scale-space for AKAZE, the diffusion is more pronounced along the directions where there is less variation of pixel intensity. This means that information across edges in the image is not mixed, thus maintaining and enhancing them.

Figure 3.7 shows an example. There is a clear edge in the image, to the right of which there is a gradient. The diffusion at the dot will be bigger along the edge, slightly smaller towards the gradient direction and the smallest towards the edge direction. As a result, the dark value in this point will not be mixed with the light value to the left of the edge. The gradient will become more gradual while the edge will remain untouched.

The mathematical details are not relevant for the application. As such, the interested reader is referred to Alcantarilla et al. [2013].

3.2.2. Scale-space

AKAZE uses a total of O octaves, number from 0 to $O-1$. Each octave as S sub-levels, from 0 to $S-1$. The scale of each image used for the scale-space is defined as

$$\sigma_i(o,s) = 2^{\frac{o+s}{S}} \tag{3.1}$$

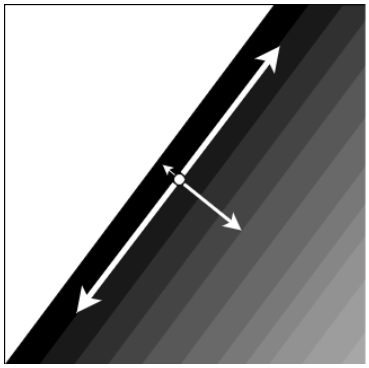


Figure 3.7: Example of relative strength of anisotropic diffusion in an image for different directions.

where σ_i is the scale of image i; $o \in [0, O-1]$ is the octave number associated to the image in question; and $s \in [0, S-1]$ is the associated sub-level number. The scale is given in pixels. Both the total number of octaves, O, and of sub-levels S, have a default value of 4 and can be changed by the user.

The scale-space is generated using Gaussian blurring to reduce artifacts and noise in the image, followed by the new method developed to increase computation speed: Fast Explicit Diffusion (FED). For each octave, the image is downsampled by a factor of 2, thus reducing the image size to half. This method is explained in detail by Alcantarilla et al. [2013]. The right image set of Figure 3.6 is an example of the resulting images.

3.2.3. Feature Detection

The measure used by AKAZE of the amount of information provided by a given pixel is the determinant of the Hessian matrix (discriminant) of the pixel values. The Hessian matrix of a continuous 2D function f is given by

$$\mathbf{H}(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \tag{3.2}$$

and its determinant by

$$\det(\mathbf{H}) = \frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial x \partial y}\right)^2 \tag{3.3}$$

This matrix describes the curvature of a function along any given direction, and the discriminant quantifies this. The higher the discriminant, the higher the curvature of the function around the point where it was calculated. In the context of an image, this would correspond to a higher contrast between the pixel and the surrounding ones.

Since the value function describing the image is discrete, some approximation is inherent in this calculation. The discriminant is thus computed through

$$L^i_{Hessian} = \sigma^2_{i,norm}\left(L^i_{xx}L^i_{yy} - L^i_{xy}L^i_{xy}\right),$$

where

$$\sigma_{i,norm} = \frac{\sigma_i}{2^{o^i}},$$

is the normalised scale factor; and L^i_{jk} are the second order derivatives of image i of the scale-space, obtained by appropriate filters.

Next, the maxima are found using the following method:

- For every scale, search the pixels for which the discriminant is above a given threshold and is the highest value in a 3×3 px window
- For each pixel found, compare it with the pixels in the adjacent scales j within a window of size σ_j
- For the maxima found, their sub-pixel and sub-scale position is determined through interpolation

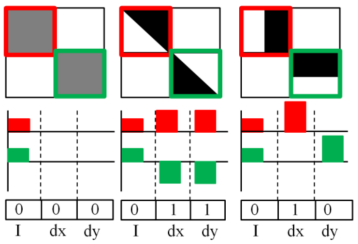


Figure 3.8: Example of tests performed in three different images with a 2×2 grid between the top left and bottom right sub-divisions. [Yang and Cheng, 2012]

- For the scale of each maximum and the two adjacent ones, fit a 2D quadratic function to the discriminant, and find the sub-pixel position of the three maxima (one for each scale)
- Fit a parabola to the three maxima, and determine the scale for which the fitted parabola has a maximum

3.2.4. Feature Description

To match the features with the ones found in other images, each feature needs a descriptor that can be compared. The method used in AKAZE is the Modified Local Difference Binary (M-LDB).

The first step is to find the dominant orientation, which allows the descriptor to be rotation-invariant. The process used is the same as in SURF [Alcantarilla et al., 2012]. This orientation corresponds to the strongest direction of the gradient of the pixel values in a 6σ radius of the feature point. Details on how this direction is determined are given by Oyallon and Rabin [2015].

The orientation is used to place a $n \times n$ grid around the point, where $n = 2$ for the first grid and increases for finer results [Yang and Cheng, 2012]. For each grid and each pair of sub-divisions three binary tests are perfomed. The tests are for intensity average, and intensity gradient in the (oriented) x and y-directions. The result is 1 when a given quantity of the first sub-division is higher than in the second one; and 0 otherwise.

Figure 3.8 shows an example of the three tests applied in three different cases in a 2×2 grid. The top left and bottom right sub-divisions are compared. The first bit corresponds to average intensity; the x- and y-direction gradients are represented by the second and third bit, respectively. In all cases the average is the same for both subdivisions. In the first case, there is no gradient in any direction. The second case as a positive gradient in both directions in the first subdivision and negative in the second (thus the corresponding bits equal to 1). In the third case, the x-gradient is positive in the first subdivision and null in the second (resulting in a bit equal to 1); whereas the opposite happens in the y-direction (yielding a bit equal to 0).

The descriptor can be constructed to three different sizes, namely 486 (full-size), 256 or 64 (light-weight) bits.

3.3. BRISK

3.3.1. Scale-space

The scale-space used in BRISK is made up of n octaves (represented as c_i, $i \in \{0, 1, ..., n-1\}$) and n intra-octaves (represented as d_i), with $n = 4$ being the default value used.

The original image provides the first octave (represented as c_0). Subsequent octaves are obtained by downscaling the previous one by a factor of 2. In other words, the image size is reduced to half. The first intra-octave, d_0, is obtained by downscaling c_0 by a factor of 1.5, meaning that d_0 is the scale in the middle of c_0 and c_1. The remaining intra-octaves are generated as were the octaves (reducing the image size to half).

3.3.2. Feature Detection

The measure of information quantity used in BRISK is called saliency, s. This value is calculated using the FAST 9-16 mask. Figure 3.9 shows the pixel adjusted circles used as FAST masks. The central red square is the central pixel (where the mask is being applied). Each different color around this pixel corresponds to a different mask. The outer one (in red) is the 9-16 mask; the blue one is the 5-8 mask. For these two masks, the numbering of the pixels is shown.

The 9-16 mask corresponds to requiring that nine consecutive pixels of the 16 in the pixel adjusted circle

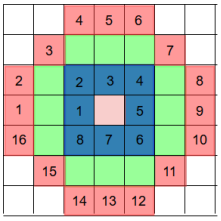


Figure 3.9: Pixel adjusted circles used as FAST masks. Numbered are the 9-16 (red) and the 5-8 (blue) masks.

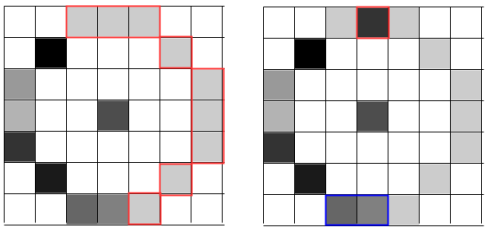


Figure 3.10: Example of corner detection in BRISK. Left - positive detection; right - negative detection.

around the central point are either brighter or darker than the central pixel by a pre-defined threshold. Figure 3.10 shows how this principle is applied with two examples. On the left, the pixels framed in red show nine consecutive pixels that are visibly brighter than the central pixel. This pixel would be recognised as a feature.

The case on the right, however, would not. The difference between the two is the top middle pixel (framed in red), which is darker than the central pixel and breaks the sequence that was used in the left case. Even though the pixels framed in blue are brighter than the middle pixel, the difference is not sufficient (this would depend on the threshold defined).

Using this method, each pixel is given a saliency score, s, defined as the maximum threshold for which that pixel is detected. In the example of Figure 3.10, the left case would have an higher saliency score than the right one.

Next, as with AKAZE (Subsection 3.2.3), the maxima are chosen. The process is represented in Figure 3.11.

In this case, the window for which a potential maximum is compared to in the adjacent scales is always 3×3 px (as shown in Figure 3.11). When searching around a pixel in c_0, a lower intra-octave, d_{-1}, is simulated by applying a FAST 5-8 mask (blue mask in Figure 3.9) to c_0. In this case, it is not required that the salience in c_0 be higher than the salience in d_{-1} for the maximum to be accepted.

The interpolation is done as in AKAZE with the difference that the s score is used instead of the discriminant. The resulting points are shown for each scale evaluated as red dots in Figure 3.11. On the right side of the same figure, the interpolation to find the sub-scale position of the maximum is shown. This process is, once again, analogous to the one used in AKAZE.

Finally, the maximum location in the space dimensions is determined by interpolating the position, shown by the red circle between intra-octave d_{i-1} and octave c_i, which lies in the line connecting the maxima points of these two layers.

3.3.3. Feature Description

The BRISK descriptor is a binary vector of 512 bits, obtained by concatenating the result from brightness comparison tests. The first step is to sample a given number of points according to a sampling pattern. Using the sampled points, two sets of point pairs are created. Based on these pairs, an orientation is determined. Finally, the descriptor is constructed based on tests performed on the pairs after rotation to fit the orientation.

The pattern shown in Figure 3.12 is used to define the sample points (represented by the blue dots arranged in circles) for each feature point found (blue dot in the centre). To avoid alliasing (a distortive effect that can occur when down-sampling images), the image is smoothed with a Gaussian filter around each sample point. The standard deviation is adjusted so that information from adjacent sampling points are not blurred in. The red circles in Figure 3.12 show the image area being smoothed around each point (note that none of them intersect). This ensures that information on a given pixel is not included in two different sample points.

From the complete set of possible pairs between the sample points, two subsets are defined: the short-distance pairings, $\mathcal{S}$, and the long-distance pairings, $\mathcal{L}$ (where $\#\mathcal{L} = L$). The short-distance pairs are all for

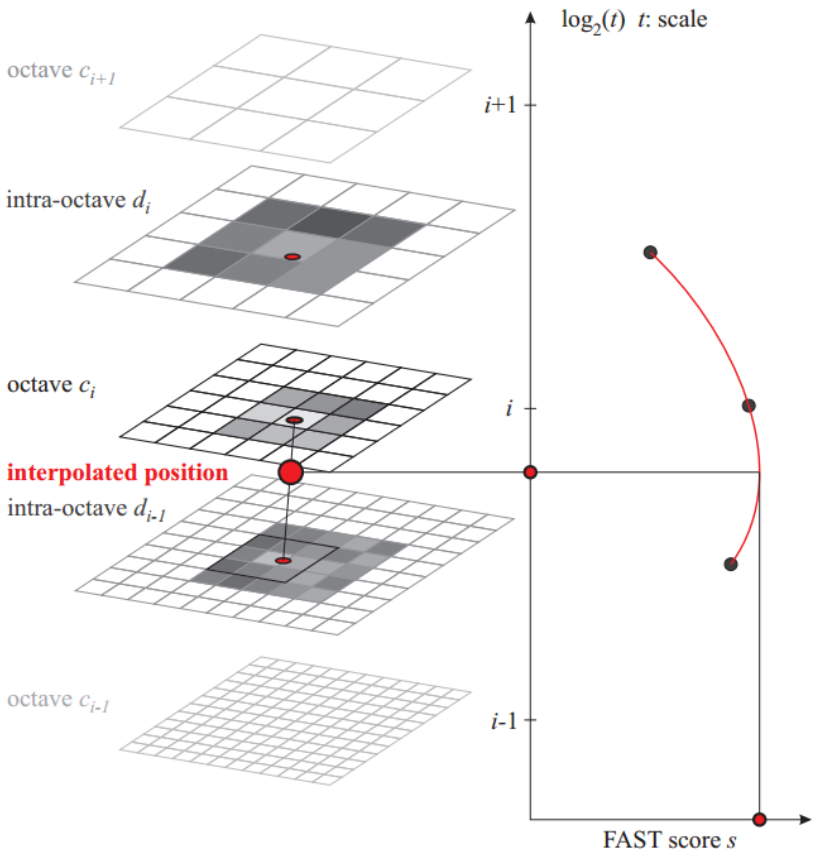


Figure 3.11: Method to determine maxima in BRISK. [Leutenegger et al., 2011]

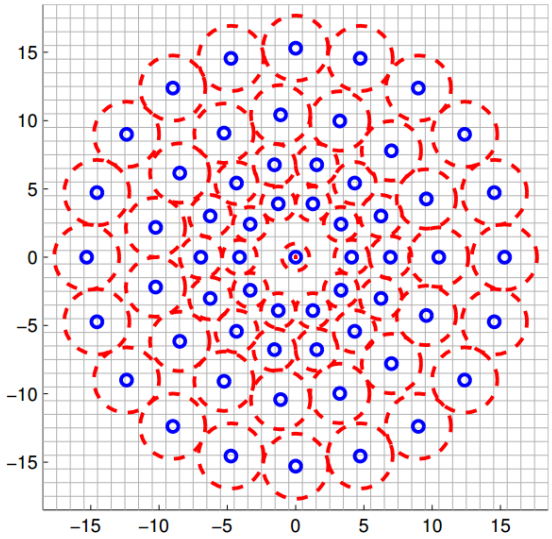
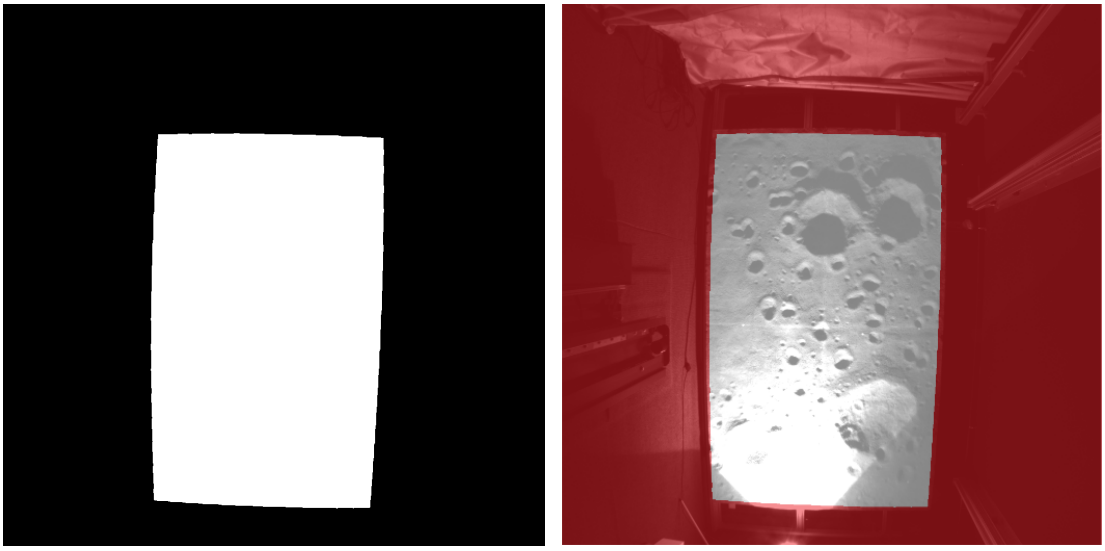


Figure 3.12: Sampling pattern for BRISK. In blue: sampling points. In red: scope of smoothing. [Leutenegger et al., 2011].



(a) Mask (b) Mask overlaid on image

Figure 3.13: Example of a feature mask used for image 19 of the nominal trajectory of dataset I

which the distance between the two points is lower than 9.75σ; the long-distance pairs require a distance greater than 13.67σ.

The orientation is determined through the local gradient of long-distance sampling-point pairs. For each pair in $\mathscr{L}$, a vector pointing from one point to the other is associated. The magnitude of the vector is proportional to the intensity difference between the two points and inversely proportional to their distances. Adding all these vectors provides the direction used to define the orientation.

Finally, the descriptor bits are defined according to intensity comparisons between points in the short-distance pairs. First, the pattern is rotated to be aligned with the orientation determined. New sampling points are then considered for the tests. A bit is 1 if the first point of the associated pair has higher intensity than the second point; and 0 otherwise.

3.4. Feature Masks

When dealing with real images, it might not be desirable to find features in the entire area of the image. One example is the consistent presence of the lander's structures in the CE-3 landing images. These structures are likely to generate strong features which may have a significant negative impact on the navigation precision.

To eliminate certain areas from the search space, masks can be used. Masks are black and white images of the same size and resolution as the image where features are to be detected. The black areas are ignored by the detector.

Masks have been used in two cases: for the CE-3 structures and horizon line; and with datasets I and II. In this second case, since some laboratory objects are visible in the images and can result in the detection of features (as observed in Subsection 6.5.7), the masks were used to limit the detection area to the terrain model. An example of a mask used in this context is shown in Figure 3.13a, which has been overlaid in the corresponding image in Figure 3.13b.

The mask used for the CE-3 images was made by composing simple shapes using the image editor *paint*; whereas the masks used in the second case were first rendered in Blender with a specific model. The masks used for the online images in datasets I and II were also distorted to match the original images captured in the lab. The full process is explained in Appendix A. For consistency, the masks should be used for both the generation of the database and the navigation solution.

3.5. Limiting Features

In practice, as explored in Section 7.1.1, the number of features of the database and used as queries for feature matching (Section 3.6) can have a significant impact on the computation time required. Furthermore, and in addition to the distribution of the features in the image, it can also affect the accuracy of the navigation

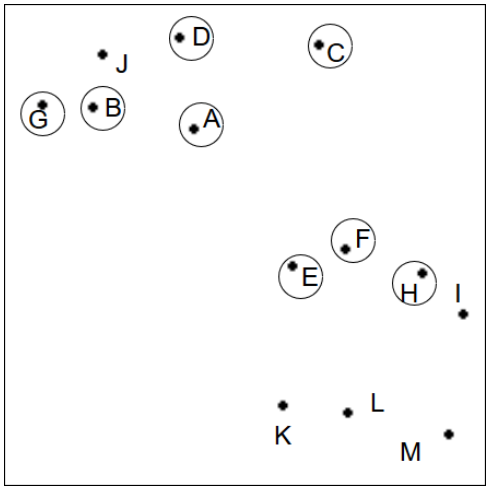


Figure 3.14: Example of feature selection for $n_{div} = 1$

solution. As such, before the matching is performed, the features are pre-selected.

If the user specifies a limit to the number of features to be considered, n_{lim} , and the number of points detected is higher than this limit, this block is run. Otherwise, all the points are taken. The pre-selection is based on the response associated to the feature, consisting of the score value used for feature detection (*e.g.*, saliency score for BRISK). The best features (higher response values) are taken.

Consider an image in which features A through M (ordered by the response value) were found, shown in Figure 3.14. If the user specifies $n_{lim} = 8$, features A through H will be selected, since these are the ones with the highest response value. The remaining features will be ignored.

Additionally, to potentially improve the accuracy of the navigation solution, the image is sub-divided into n_{div} even rows and columns. An example of this is applied to the same image in Figure 3.15, where $n_{div} = 2$. The sub-divisions are numbered from zero to $n_{div}^2 - 1$ starting from the top left corner and increasing first in the horizontal direction (marked in the bottom right corner of each section). In this case, there are 4 sections and the number of features per section – $n_{f/div}$ – is two (eight features equally divided among the sections).

First, the feature with the highest response value, A, is taken. This feature belongs to the image section 0, so it is placed at the first slot of the corresponding list. The next feature, B, belongs to the same section. Thus, it is placed in the second slot of the same list. Since feature C also belongs to section 0 and the associated list is already full, it is placed in the first slot of the remaining features (here marked with an "R"). The same happens with feature D.

The next highest value found is for feature E, which belongs to section 3. Since this list is still empty, this feature takes the first slot. This process continues until either all lists associated to image sections are filled or all features have been organised. In the end, the features from the first four lists are taken (A, B, K, E and F). These are marked by circles in Figure 3.15. The three missing features are taken from the first three slots of list "R" (C, D and G), marked with lozenges.

It is also possible to set aside a given percentage of the features to be independent of the image sub-division, *percentage non div*. The algorithm is as such:

- Create a list, *point responses*, of arrays of the form: [*response*, *division id*, *point id*, *point coordinate*, *point descriptor*], where *division id* is the image sub-division in which the feature belongs to
- Determine number of features per sub-division: $n_{f/div} = round\left(\frac{1-percentagenondiv}{n_{div}^2} n_{lim}\right)$
- Sort *point responses*
- For each point in *point responses* starting from the end (decreasing value of response)
  - If any subdivision is not full
    - ◇ If subdivision of current point is full: append point to overall list
    - ◇ Otherwise: append point to final list of pre-selected features
  - If all subdivisions are full and number of points iterated is higher or equal to n_{lim}: stop
- Append first points in overall list to final list, such that final list has n_{lim} points

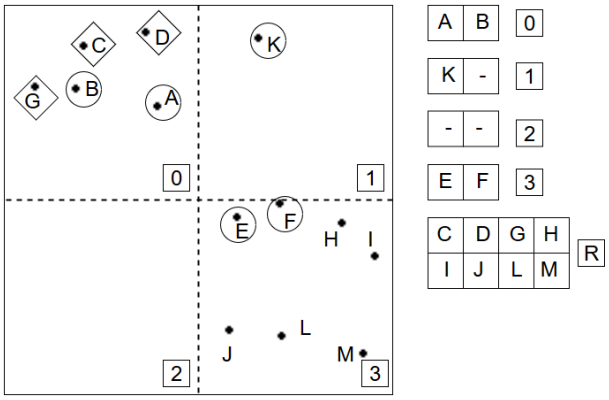


Figure 3.15: Example of feature selection for $n_{div} = 2$

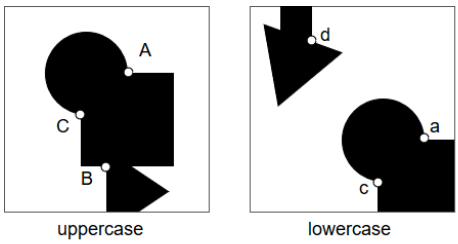


Figure 3.16: Hypothetical feature matching example

3.6. Feature Matching

As explored in Sections 3.2 and 3.3, for each given image, a feature detection algorithm finds the features within it and prescribes them a descriptor. This descriptor is what characterises a feature. Looking at Figure 3.16, the descriptor used to characterise feature "A" will be very different from the used for feature "B". On the other hand, features "A" and "a" will have very similar (if not equal) descriptors.

This property (the repeatibility of the feature descriptor) can be used to match features between images. If the descriptors are thought of as points in a k-dimensional space (where k is the size of the descriptor, *e.g.*, 512 for BRISK), the closer two descriptors are to one another, the more similar they will be.

As such, a good method to match features is through a NN search. Given the high number of features that can be found in an image and the size of each of their descriptors, it is not practical to find the solution through a linear search (compare a descriptor with all its possible matches). This would not be adequate for real-time usage of the application.

Other alternative faster methods exist, which still have high performance. One class of such methods, used for this application, organises the descriptors in the form of trees (indeces) that are later used to perform hierarchical searches.

It is to note, however, that all features will have a pair assigned to them, regardless of how similar their descriptors are. For the example in Figure 3.16, if the features in image "uppercase" were being matched to the ones in image "lowercase", a pair would still be made with feature "B". In this case, the matches "A"-"a" and "C"-"c" would be expected. Considering the hypothetical pair "B"-"d", the distance between their descriptors would be much higher than the one for either of the correct matches. This assumption will allow to improve the percentage of correct matches used to calculate the camera pose.

A brief introduction to these indeces, their construction and the search algorithms involved is given in Subsection 3.6.1. The OpenCV library FLANN, which is explored in Subsection 3.6.2, allows for the optimisation of these indeces in terms of build time versus search time and time versus memory overload. The optimised indeces thus obtained can also be saved and loaded for later use. In Section 3.7 it is explained how the matches used for the pose calculation are chosen.

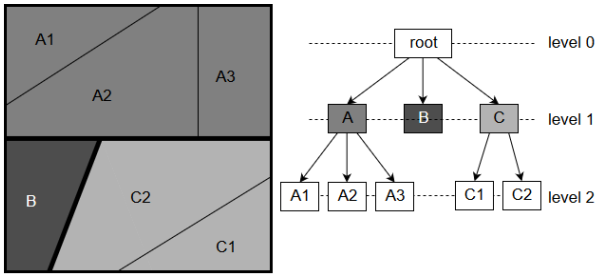


Figure 3.17: Example of a 2-d tree

3.6.1. Indices for Nearest Neighbour Searches

The indexes used in approximate NN searches can also be called k-d trees. A k-d tree is a data structure that organise k-dimensional (k-d) points in the form of nodes, branches and leaves. The root (the first node) corresponds to the whole set of data. This set is, then recursively divided into smaller sets branching out of the original one into new nodes. The terminal nodes are the leaves. The logic used to divide each subset and the stopping criteria is what distinguishes each algorithm.

Figure 3.17 shows an example of this structure for a 2-d space. At level 0 is the root, corresponding to the complete data domain. The root is divided into three nodes, represented at level 1 by A, B and C. Then, nodes A and C are further divided into A1, A2 and A3, C1 and C2, respectively, which belong to level 2 and are the leaves, along with node B. Note that the leaves need not be in the same level, nor is there necessarily a fixed number of divisions for each node.

Since FLANN (Subsection 3.6.2) both chooses the best structure for a given dataset (index optimisation), builds the index and performs the search, an indepth understanding of how this is done is not required to comprehend the proposed application. Regardless, to show how the search time can be significantly improved while mantaining very high matching precision, one algorithm for index construction and searching will be explained, namely the hierarchical K-means. Other algorithms available include the multiple randomised k-d trees, which is described in detail by Muja and Lowe [2014].

Hierarchical K-means Trees

This method results in partitioning the dataset provided into Voronoi cells: regions in the k-d space for which every point contained within it is closer to the cell's centroid than to any other centroid. This concept is made clearer by the process in which the cells are defined. The construction can be varied by changing two parameters, namely the branching factor, K, and the number of iterations allowed during the cell division. The branching factor is the number of divisions for each node.

Figure 3.18 shows how this process is applied at the first level of a 2-d dataset with $K = 3$. The 2-d space represented has coordinates k_1 and k_2, corresponding to the first and second entries of a vector in the space, respectively. There are seven points in the space, represented by dots.

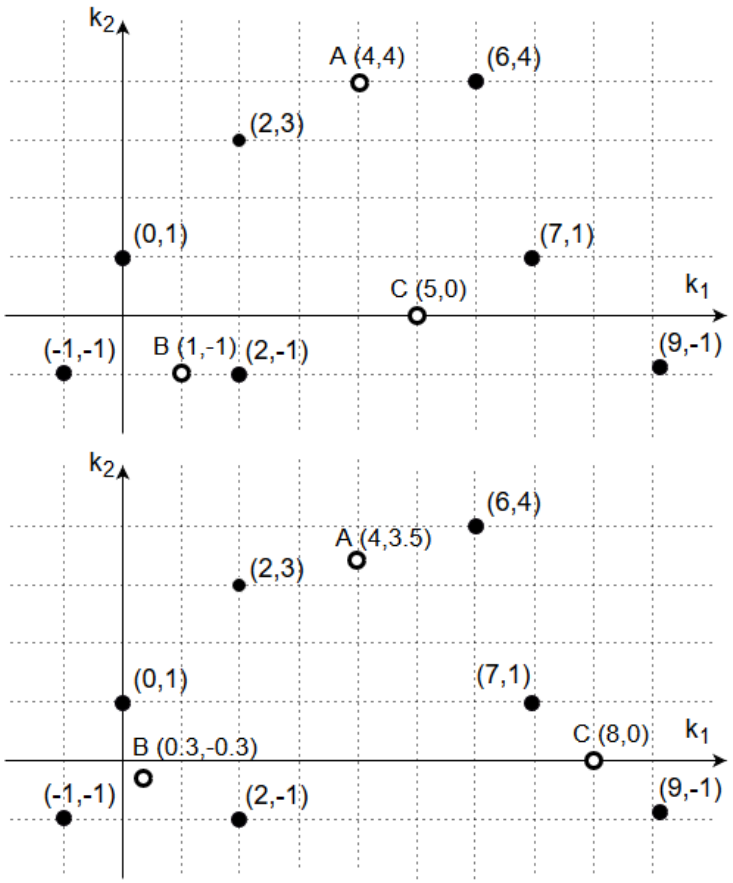
First, three (from the K parameter) centroids are placed at random in the 2-d space. Then, as given in the top tables in the figure, the distances of the points to each centroids are calculated. For each point, the centroid for which the smallest distance is observed is chosen (last column of top table). For example, point $(2,-1)$ is closer to centroid B (at a distance of 1.0) than to centroids A or C (at distances of 5.4 and 3.2, respectively).

The coordinates of the new centroids, A', B' and C', are calculated based on this distribution. For example, points $(7,1)$ and $(9,-1)$ were the points assigned to centroid C (at $(5,0)$). The position of C' (the new centroid to be used in the next iteration) is the centroid of these two points, therefore $(8,0)$.

This process is repeated until convergence. In this case, the convergence was observed after one single iteration. Once a node has been divided, each of the resulting sub-nodes (group of points associated to each of the final centroids) are divided in a similar manner. The final tree constructed from this dataset can be seen in Figure 3.19. Since nodes A and C have less than three (K) points, they require no further division and are leaf nodes. Node B has three points, thus each of them becomes a centroid in the second level division.

The method is as follows:

1. Define K random points as the centroids of the new K nodes
2. Calculate distance of each data point to each centroid and associate them to the closest one, thus defining K sub-sets (one for each centroid)



Distances	A	B	C	centroid
(0,1)	5.0	2.2	5.1	B
(-1,-1)	7.1	2.0	6.1	B
(2,-1)	5.4	1.0	3.2	B
(2,3)	2.2	4.1	4.2	A
(6,4)	2.0	7.1	4.1	A
(7,1)	4.2	6.3	2.2	C
(9,-1)	7.1	8.0	4.1	C

New centroids		
A' (4,3.5)	B' (0.3,-0.3)	C' (8,0)

Distances	A	B	C	centroid
(0,1)	4.7	1.4	8.1	B
(-1,-1)	6.7	1.5	9.1	B
(2,-1)	4.9	1.8	6.1	B
(2,3)	2.1	3.7	6.7	A
(6,4)	2.1	7.1	4.5	A
(7,1)	3.9	6.8	1.4	C
(9,-1)	6.7	8.7	1.4	C

New centroids (converged)		
A' (4,3.5)	B' (0.3,-0.3)	C' (8,0)

Figure 3.18: First division of a 2-d dataset using $K = 3$; dots – data points; circles – centroids

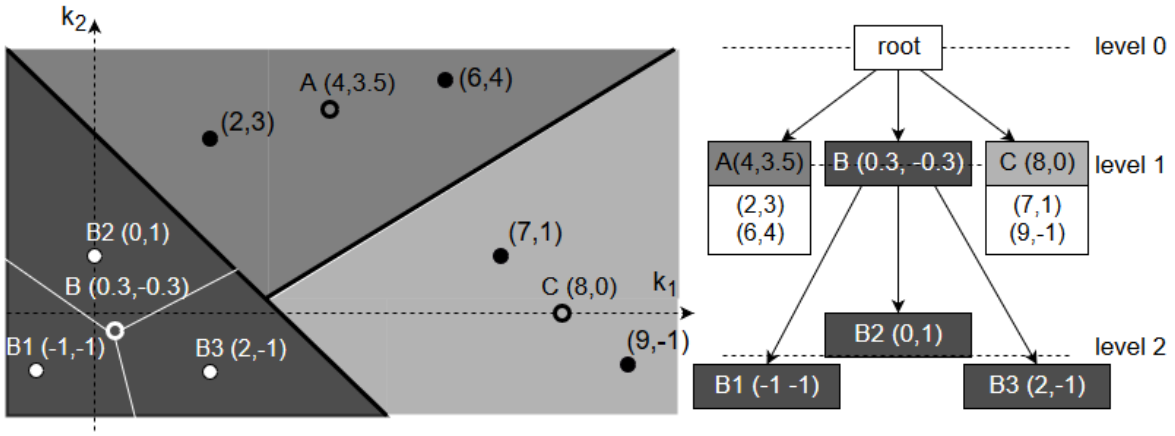


Figure 3.19: Hierarchical tree constructed from 2-d dataset and $K = 3$

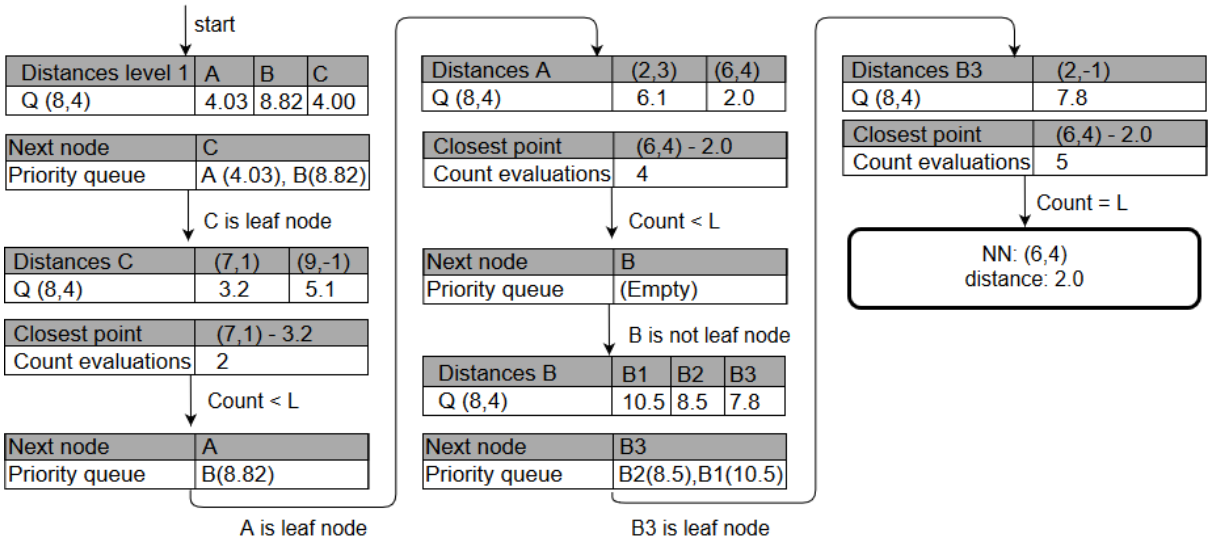


Figure 3.20: Diagram of search through hierarchical 3-means tree for $Q = (8,4)$

3. Calculate the new centroid of each subset, using the data points assigned to it
4. Repeat 2. and 3. until convergence or maximum number of iterations reached

Now that the tree has been built, it can be used for any number of NN searches. Each search is limited to L evaluations, meaning it stops once the distance to L points of the dataset have been calculated.

Figure 3.20 represents the search of query point $Q = (8,4)$, using the tree shown in Figure 3.19 with $L = 5$.

The process starts at the root, which contains three sub-nodes, corresponding to the level 1 of the tree. The distance of Q to each of the nodes' centroids is calculated: *e.g.*, for node A the point used is $(4,3.5)$. The results are shown in the first table. Node C has the centroid closest to Q, at 4.00 distance, therefore it will be the next node evaluated. Nodes A and B are ordered according to their distances in the priority queue: first A, with a distance of 4.03; then B, with a distance of 8.82.

Since C is a leaf node, the distance to the points contained within it (points $(7,1)$ and $(9,-1)$) is evaluated. The closest point, $(7,1)$, is saved. The evaluation count (2) is lower than the limit ($L = 5$), so the search proceeds. The next node in the priority queue is checked, in this case, node A. Since it is also a leaf node, the same process as with node C is repeated. Here one of the points was found to have a smaller distance to Q than the point saved from node C, namely point $(6,4)$ at a distance of 2.0 (less than 3.2 found for point $(7,1)$).

At the end of this evaluation, the evaluation count is still lower than the limit, so the next node is selected from the queue: node B. This node is not a leaf node, so the process taken at the root is repeated. In other words, the distance to the centroid of each sub-node of node B is determined. The one for which the distance is smallest is used to continue the search (in this case, B3) and the remaining ones are ordered in the priority queue according to their distance (first B2, then B1). After evaluating node B3, the evaluation count reaches the limit (5) and the search stops and the last saved point, $(6,4)$, is taken as the solution.

For bigger datasets and smaller relative search limit values, this method can greatly reduce the computational requirements in comparison with the linear search.

The method used to search for the NN given a query point, Q, is as follows:

1. Start at root
2. If current node is leaf node
   - Calculate distances in all points within node
   - Choose point for which minimum distance was found
   - Increase count by number of points evaluated
   - If count is equal to or above L: stop; otherwise repeat from 2.
3. Otherwise
   - Calculate distances to centroids of sub-nodes
   - Pick node for which distance is smallest
   - Order remaining nodes in priority queue by increasing distance
   - Repeat from 2.

3.6.2. Fast Library for Approximate Nearest Neighbours (FLANN)

As previously mentioned, FLANN is a library incorporated in OpenCV which allows for constructing indexes with parameters provided by the user (*e.g.*, what tree algorithm to use); searching the k NN (the k data points closest to the query ones) through an index; and index optimisation.

This automatic optimisation finds the best tree structure for a given dataset, sampling fraction (fraction of dataset used for optimisation) and search precision. To do so, various sets of parameters are chosen in two stages: first through a grid method covering all possibilities; then, the best solution is refined using an heuristic algorithm (does not guarantee finding a solution, but produces useful results in practice). For details, the reader is referred to Muja and Lowe [2014].

The cost function for the optimisation is

$$cost = \frac{s + w_b b}{(s + w_b b)_{opt}} + w_m m, \tag{3.4}$$

where s is the search time; b the tree building time; w_b the relative weight of the build-time with respect to the search time ($w_b = 0$ results in the algorithm for which the search time is minimum); m is the memory overhead; w_m is the relative weight of the memory overhead with respect to the time overhead; and $(s + w_b b)_{opt}$ is the optimum time cost (result if memory overhead is not taken into account).

Since, for the application in question, the tree is built offline, there is no restriction on the build time, thus $w_b = 0$. During the initial study of the application, a requirement on the memory load will not be set, thus $w_m = 0$. However, this parameter could be ajusted in a trade-off between memory load, on-board computational time and navigation precision.

Once an index is built, either by setting the parameters or by optimisation, it can be saved in a *dat* file. To later reuse the index, the dataset used to generate it should also be saved (which will be done in the form of a *npy* file).

3.7. Match Downselection

Even though often used for that purpose, FLANN is not an algorithm to find matches between features. Rather, it always provides a solution for every query point given, regardless of the distance between it and the associated solution. In other words, it does not provide any protection against incorrect matches. On the other hand, it can yield the same solution for multiple query points. Meaning, the same data feature can be matched to two different query features.

This last phenomenon could be physically valid. For example, the same image structure could be detected at two different scales in the same image, resulting in two different query features for the same physical point. If in the index only one data feature exists for that point, then it would indeed be the correct match for both queries. However, this could not be handled correctly by the pose determination algorithm (see Chapter 4.4).

As such, it is important to perform a pre-selection of the matches: first, by eliminating duplicate matches; second, by choosing the N best matches. This second selection will allow to decrease the percentage of incorrect feature matches (as discussed in the beginning of the chapter).

The downselection algorithm was changed after the time distribution analysis discussed in Section 7.1.1. The first algorithm separated the two operations (finding the unique matches and selecting the best ones). The selection of the best matches was very time consuming, being the determining factor for the complete pose determination. This motivated joining the operations. The new algorithm takes advantage of the way python sorts lists of arrays (the list is ordered according to the values of the first entry of the arrays, using the second entry to solve ties, and so forth), as was also done for the feature pre-selection algorithm (Section 3.5).

The improved algorithm is as follows:

- Initialise list, $best\_similarities$, of size equal to the number of points used to build the index with $[max_d + 100, ii, -1]$, where max_d is the maximum distance found among the FLANN NN solutions; and ii the index of the list entry
- For every query point, $\mathbf{p}_{query}$, get id_{data} of the data point it was matched to
  - If the first element of entry id_{data} of $best\_similarities$ is higher than the query point's distance, d_{query}: set the entry id_{data} of $max\_similarities$ to $[d_{query}, id_{data}, \mathbf{p}_{query}]$
- Sort $best\_similarities$ and pick first N entries, where N is the number of matches specified

Table 3.1: First three and sixth entries of *point response* list after being created

response	*division id* ($n_{div} = 1$)	*point id*	*point coordinate*	*point descriptor*	*division id* ($n_{div} = 2$)
0.001055	0	0	(307.0, 262.1)	[0, 0, 1, ..., 1, 1, 1]	0
0.000644	0	1	(357.1, 262.0)	[0, 0, 1, ..., 0, 1, 1]	0
0.000822	0	2	(272.7, 263.1)	[0, 0, 0, ..., 0, 0, 0]	0
0.000586	0	5	(592.1, 270.3)	[0, 1, 1, ..., 0, 1, 0]	1

Table 3.2: First three entries of *point response* list after being sorted

response	*division id* ($n_{div} = 1$)	*point id*
0.0005003	0	964
0.0005009	1	29
0.0005010	0	84

3.8. SUMMARY EXAMPLE

To facilitate the understanding of the algorithms introduced in this chapter, an example is given with two data images (numbers 19 and 23) and one online image (number 22) from the nominal trajectory of dataset I (described in Subsection 7.2.1). First, the database will be generated using the two data images. Then, the features found in the online image will be matched to the database features. Despite being applied in both cases, the effect of the mask will be shown for the online image, since it is more evident for it.

Two databases were generated, with n_{div} (see Section 3.5) set to 1 and 2, respectively. In both cases, n_{lim} was set to 500. The first step to build the database is to detect the features in the data images, which was done with AKAZE. The total number of detected features was 1737 and 2329 for data image 19 and 23, respectively.

Next, the features are limited to 500 (n_{lim}), with the algorithm described in Section 3.5. Table 3.1 shows the first three and the sixth entries of the list *point responses* when it is created, for image 19 with $n_{div} = 1$. The table also shows (in the last column) the subdivision to which the point was assigned to for the case with $n_div = 2$. Note that for $n_{div} = 1$, all points have *division id* equal to 0, whereas, for $n_{div} = 2$, the sixth entry belongs to subdivision 1. Also, the *point id* always increments by 1, as expected.

Table 3.2 shows the first three columns of the first three entries for the same image with $n_{div} = 2$. Note that the *division id* entry is no longer in increasing order. Instead, this is viewed in the entry *response*, showing that the sorting is successful.

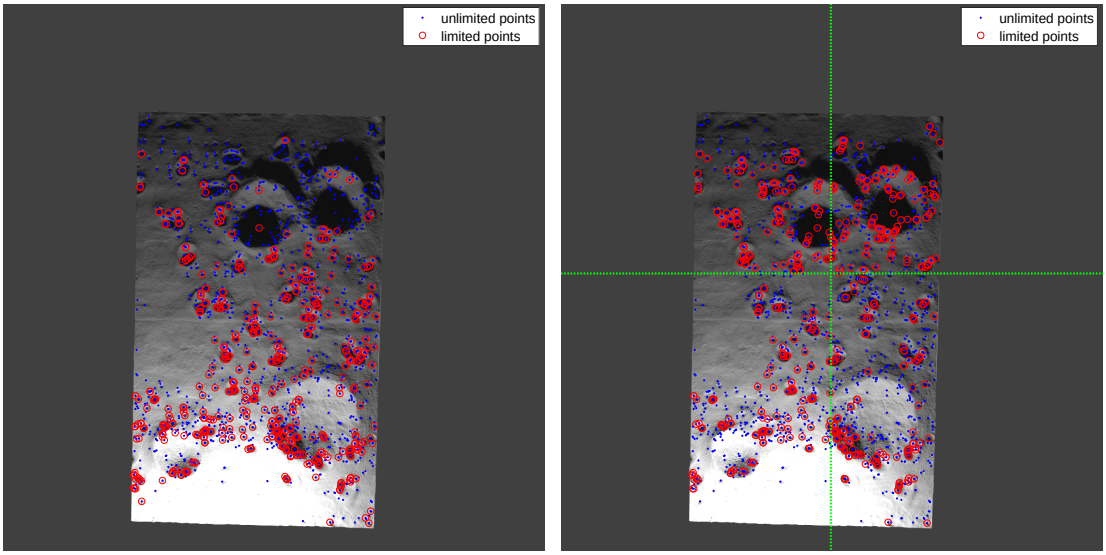
Once the list has been sorted, they are limited taking into account the image division. For $n_{div} = 1$, the last 500 points of the list (for which the *response* is highest) are chosen. For $n_{div} = 2$, the image division plays a role. This case is observed next.

The image is subdivided into four sections. The number of features to be assigned per division is $n_{f/div} = 125$. A list *count* with five entries (four for the subdivisions and one for the remaining points) is initialised with values 0. The search starts with the last point in the list *point responses* and the division in which it belongs is checked. In this case, the point belongs to subdivision 3 and is added to the associated list. The fourth entry of *count* was incremented to 1.

Table 3.3 shows iterations number 261 to 265 of the process. The position of the point in the *point response* list is shown in column "Point id". As expected, this number is decreasing. The next column shows the subdivision to which the point belongs to, followed by the current associated value stored in the *count* list. The last column shows the list to which the point is added, where lists 0 to 3 correspond to the subdivisions, and "R" to the remaining points.

Table 3.3: Feature limiting process for iterations 261 to 265, for image 19 with $n_{div} = 2$

Iteration number	Point id	*division id*	*count[division id]*	List chosen
261	1478	3	118	3
262	1477	2	125	R
263	1476	2	125	R
264	1475	0	12	0
265	1474	3	119	3



(a) $n_{div} = 1$ (b) $n_{div} = 2$

Figure 3.21: Features found and selected in image 19

For iteration 261, the point belongs to subdivision 3, which has 118 points included in the associated list. Since this list still doesn't have the 125 points required ($n_{f/div}$, the new point is added to it. The next two points belong to subdivision 2, which already has 125 points. Thus, they are added to the "R" list. The point found in iteration 264 belongs to subdivision 0, which only has 12 points, so the point is added to the associated list. In iteration 265, the point belongs to subdivision 3. It can be seen that the number of points in list 3 was successfully incremented after iteration 261.

In this case, the process stopped when 125 points were found in all subdivisions and no point from list "R" was added.

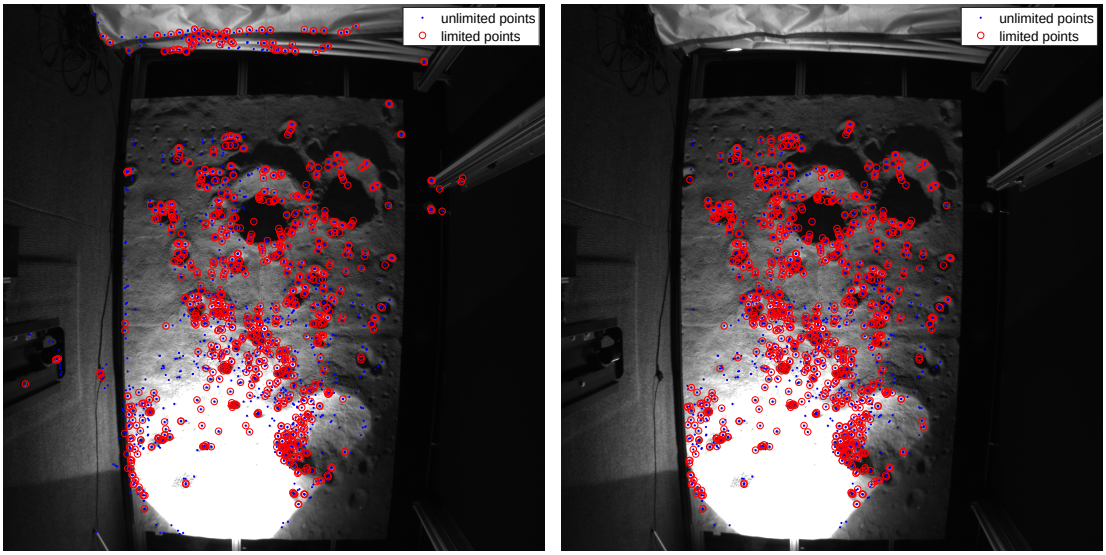
Figures 3.21a and 3.21b show the features found in image 19 with $n_{div} = 1$ and $n_{div} = 2$, respectively. The blue dots represent all the features found (a total of 1737) and the red circles represent the features selected with the limiting algorithm exemplified above. The green lines in Figure 3.21b represent the divisions made. The example clearly shows the effect of enforcing the image division on the feature selection: using $n_{div} = 2$ results in a more even distribution of the features throughout the image area, with much more features in the top right corner of the model than for $n_{div} = 1$.

The database can now be built with the selected features. The associated descriptors are saved and are organised into an index (using FLANN). The structure of this index cannot be viewed directly and is not relevant for the application. Apart from these two elements, the database also includes the world coordinates associated to each of the features. This element will be discussed in Subsection 4.3.

With these database elements, it is now possible to find and down-select matches to features in an online image. For the remainder of the example n_{div} will be set to 2. To show the effect of using feature masks, the features were detected and selected with and without the use of the appropriate mask. The result is shown in Figure 3.22. It is readily clear that the mask was important to limit the features to the relevant area, since in Figure 3.22a many were detected for objects of the laboratory, such as the curtains.

Next, the features selected are matched to the ones in the database, on the basis of their descriptors, through the FLANN index. This matching is done automatically and the details are not accessible or relevant for the application. Once the matching has been performed, the pairs can be down-selected in order to obtain the best unique matches. The quality of the match is assessed based on the difference between the descriptors of the online and database features paired. A match is unique if no other online point has been matched to the same database point.

The number of matches to take, N, was set to 50. The first step is to create the list *best_similarities* of size 1000 (number of features in the database, 500 for each of the two images). Each entry (row) contains another list with three subentries (columns). The first column is the same for all rows and is calculated based on the maximum distance found between paired points, max_d. In this case, the value was 299 ($max_d + 100$). The



(a) Without mask (b) With mask

Figure 3.22: Features found and selected in the online image 22

Table 3.4: Match down-selection process for first three points and point 35 selected for online image

Online point id	Data point id	Current distance	Saved distance	Save point
0	2	120	299	yes
1	19	119	299	yes
2	2	125	120	no
35	19	76	119	yes

second column contains the index of the list row (starting from 0 and incrementing by 1). The third column is initialised with -1.

The first four operations are summarised in Table 3.4. The first column, "Online point id", shows the index of the online point. The second column, "Data point id", shows the point in the database the online point was matched to. The "Current distance" is the difference between the descriptors of the online point considered and the matched data point, whereas the "Saved distance" is the distance found in the first column of *best_similarities* in the row associated to the matched data point. Finally, the last column, "Save point", shows whether the point was saved or not.

The first two points are saved, since they were the first online points found that were matched to their associated data points (the "Saved distance" is equal to 299, the value used to initialise the list). For the first point, row 2 of the list *best_similarities* was set to [120, 2, $\mathbf{p}_{query}$], where $\mathbf{p}_{query}$ is the image coordinates of the online feature being matched.

The third point was matched to the same data point as the first one (both were matched to data point 2). Since the distance of the third point (125) is higher than the distance found for the first point (120), which was saved in *best_similarities*, this point is discarded. On the other hand, online point 35 was matched to the same data point as the second online point (both were matched to data point 19). In this case, the distance was better for the new online point (76 in comparison to 119), so this point was kept, and online point 1 was discarded.

The process was continued until all selected features were tested. At the end, the list *best_similarities* was sorted and the first 50 entries were taken (the ones for which the matches had the most similar descriptors, and thus, the distance was smaller).

4 | Camera in 3D World

Once the relevant information has been obtained from the data and online images (discussed in Chapter 3), it has to be related to the physical world to allow for the pose determination.

Using the feature matches obtained through the methods described in Chapter 3, the camera pose can be determined, provided information of the 3D world in which the camera is placed. The required information includes a suitable model to represent the camera and the 3D world coordinates of the data features used to build the database.

Section 4.1 describes the reference systems, state variables and transformations involved. The camera model is described in Section 4.2. Section 4.3 describes the method used to calculate the real world coordinates of a point given its pixel coordinates in the image and its depth data. Finally, the algorithm used to determine the camera pose from the match between the 2D online feature coordinates and the 3D data world coordinates is introduced in Section 4.4.

4.1. Reference Systems

The reference systems and state variables used are defined in Subsections 4.1.1 and 4.1.2, respectively. In Subsection 4.1.3, the transformations between different reference systems and variables are given.

4.1.1. Reference Systems

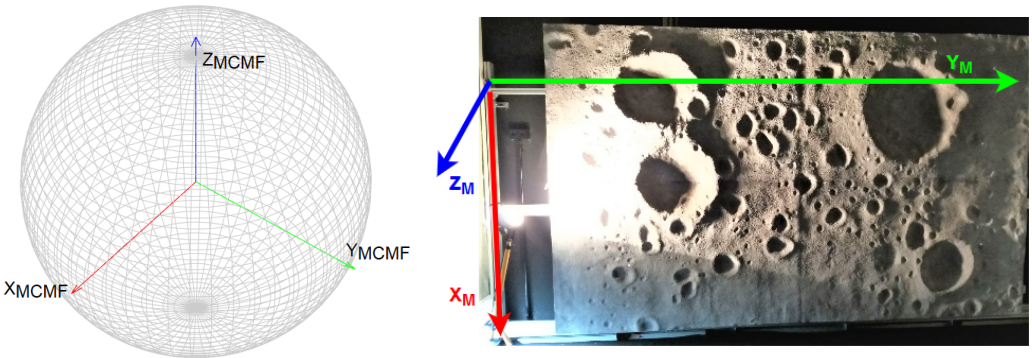
The reference systems of interest are the MCMF frame, $\mathcal{F}_{MCMF}$; the 3D model frame, $\mathcal{F}_M$; the camera frame as used in the pinhole model, $\mathcal{F}_C$, and in the 3D modeling softwares, $\mathcal{F}'_C$; the image frame, $\mathcal{F}_i$; and the depth data frame, $\mathcal{F}_z$.

Moon-Centred Moon-fixed Frame

The MCMF frame, $\mathcal{F}_{MCMF}$ is shown in Figure 4.1a. It has the origin in the Moon's geometric centre; the Z_{MCMF}-axis is along the Moon's rotational axis; and the X_{MCMF}- and Y_{MCMF}-axis are fixed with respect to the Moon's surface. To uniquely define these two axis, X_{MCMF} points towards the Earth and Y_{MCMF} completes the right-hand rule, as used for the lunar mapping done by the Kaguya mission.

This reference system is used, since the Kaguya data was used to construct the 3D lunar model and the trajectory of the CE-3 lander is given in this system.

3D Model Frame



(a) MCMF frame

(b) Terrain model frame

Figure 4.1: Model reference frames

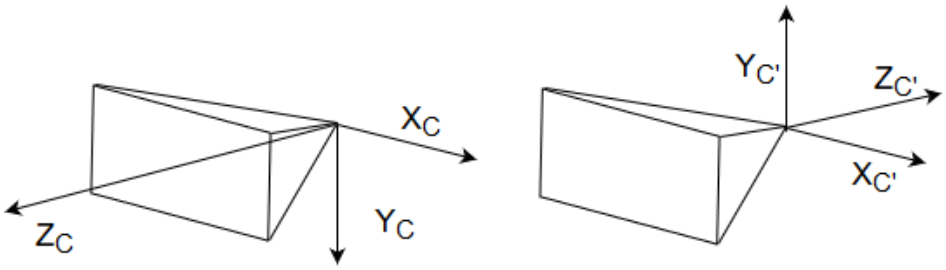


Figure 4.2: Camera frames used for the pinhole model (left) and in the 3D software environments (right)

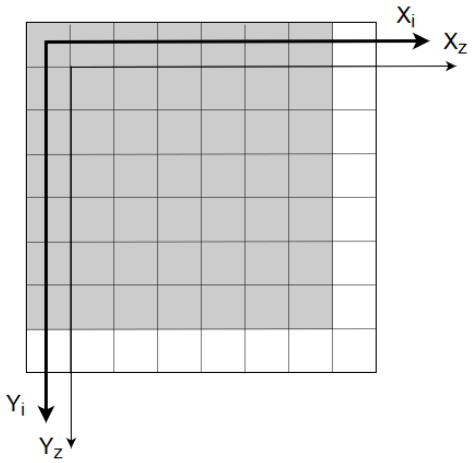


Figure 4.3: Image and depth data frames

The 3D model frame, $\mathcal{F}_M$, is used by the rendering software to place elements in the 3D environment. For the lunar model, this frame corresponds to the MCMF frame. For the TRON terrain model, it corresponds to the frame defined by the three reflectors placed in the edges of the model, and is shown in Figure 4.1b. In this case, the origin is on the top-left reflector; the X_M-axis points along the horizontal towards the top-right reflector; the Y_M-axis is in the plane defined by the three reflectors and perpendicular to the X_M-axis pointing down; and the Z_M-axis completes the right-hand rule.

Camera (Pinhole Model) Frame

The camera frame, $\mathcal{F}_C$, is used in the pinhole camera model (see Section 4.2.1) and to define the camera pose. This frame is represented in Figure 4.2 (left). The origin is at the camera's principal point; the Z_C-axis is along the principal axis pointing foward; the X_C-axis is along the horizontal side of the camera sensor pointing right; and the Y_C-axis completes the right-hand rule (along the vertical side of the sensor pointing down).

Camera (Rendering Softwares) Frame

The rendering softwares used place the camera according to a slightly different frame, $\mathcal{F}'_C$. This frame has the opposite direction of the Y- and Z-axis as in frame $\mathcal{F}_C$, as shown in Figure 4.2 (right).

Image Frame

The image frame, $\mathcal{F}_i$, is a 2D frame used to represent the pixel's positions in the image, and is shown in Figure 4.3. The origin is at the centre of the top-left pixel; the X_i-axis points right, along the horizontal side of the image; and the Y_i-axis points down along the vertical side.

Depth Data Frame

The depth data frame, $\mathcal{F}_z$, is another 2D frame that can be used to visualize the way that the depth data is organised. It is simply shifted from $\mathcal{F}_i$ by 0.5 px in both directions, meaning the origin is at the centre of the top-left pixel, as can be seen in Figure 4.3. The depth data is defined for the discrete values in this frame, meaning at the centre of each pixel. It is used to retrieve the depth of the world point associated to a given pixel (see Section 4.3).

4.1.2. State Variables

Of interest are the position and orientation of the camera with respect to the Moon or the terrain model. The position can be represented by Cartesian or spherical coordinates. The orientation can be given in various forms, of which the ones considered will be rotation cosine matrices and quaternions.

Cartesian coordinates

The Cartesian coordinates are used to place the camera in the 3D model by both the image rendering softwares considered (Blender and SensorDTM). Thus, the input camera poses will be given in these coordinates. The output from the pose determination algorithm – EPnP – is also given in this format. To facilitate the calculation of the navigation accuracy, the navigation solution will be determined in these coordinates.

They can be represented as a vector $\mathbf{p}^X = \begin{bmatrix} x^X & y^X & z^X \end{bmatrix}^T$, given in the $\mathcal{F}_X$ frame, where X refers to one of the frames defined in Subsection 4.1.1.

Spherical coordinates

The data available for the reference mission (introduced in Subsection 2.4), namely the lunar DEMs and the CE-3 trajectory, are provided in spherical coordinates. These are:

- latitude, δ, $-90° \leq \delta \leq 90°$; positive to north
- longitude, τ, $0° \leq \tau < 360°$ (used in calculations and in the DEMs) or $-180° \leq \tau' < 180°$; positive to east (used in the CE-3 trajectory data)
- radial distance: distance from the sphere's centre to a point, r; or altitude: distance from the surface of the body to a point, h (used in the CE-3 trajectory data)

To transform the CE-3 trajectory data to the format required by the rendering software and the pose determination algorithm (Cartesian coordinates), the following operations were done:

$$\begin{aligned} \tau &= \tau' + 360 \\ r &= r_{Moon} + h_{surf}(\delta, \tau) + h \end{aligned} \tag{4.1}$$

where r_{Moon} is the lunar radius and $h_{surf}(\delta, \tau)$ is the altitude of the surface point at the given latitude and longitude (retrieved from the DEM) with respect to the reference sphere (radius equal to r_{Moon}).

Transformation from Spherical to Cartesian Coordinates

To transform from the spherical coordinates (in which the CE-3 trajectory data is available) to Cartesian coordinates, the following equation was used:

$$\begin{aligned} x^{MCMF} &= r\cos(\delta)\cos(\tau) \\ y^{MCMF} &= r\sin(\delta)\cos(\tau) \\ z^{MCMF} &= r\sin(\delta) \end{aligned} \tag{4.2}$$

The opposite transformation is not required.

Rotation Cosine Matrix

Rotation cosine matrices are transformation matrices used to rotate vectors in Euclidean space and can be used to define the orientation of one reference frame with respect to another. Take the rotation matrix $\mathbf{R}_A^B$

$$\mathbf{R}_A^B = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \tag{4.3}$$

which provides the transformation from $\mathscr{F}_A$ to $\mathscr{F}_B$, two frames with the same origin. Take the unit vector of the x-direction of $\mathscr{F}_A$, $\hat{\mathbf{x}}_A$, which is given in the $\mathscr{F}_A$ frame by

$$\hat{\mathbf{x}}_A^A = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T \tag{4.4}$$

or in the $\mathscr{F}_B$ frame by

$$\hat{\mathbf{x}}_A^B = \begin{pmatrix} x_X^A & y_X^A & z_X^A \end{pmatrix}^T \tag{4.5}$$

By definition,

$$\hat{\mathbf{x}}_A^B = \mathbf{R}_A^B \hat{\mathbf{x}}_A^A \tag{4.6}$$

Thus

$$\hat{\mathbf{x}}_A^B = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} a \\ d \\ g \end{pmatrix} \tag{4.7}$$

Applying this with the remaining unit vector, $\hat{\mathbf{y}}_A$ and $\hat{\mathbf{z}}_A$, results in

$$\mathbf{R}_A^B = \begin{bmatrix} \hat{\mathbf{x}}_A^B & \hat{\mathbf{y}}_A^B & \hat{\mathbf{z}}_A^B \end{bmatrix} \tag{4.8}$$

Rotation matrices are orthonomal, thus the inverse of a matrix is given by its transpose, as follows

$$\mathbf{R}_B^A = (\mathbf{R}_A^B)^{-1} = (\mathbf{R}_A^B)^T \tag{4.9}$$

Thus,

$$\mathbf{R}_B^A = (\mathbf{R}_A^B)^T = \begin{bmatrix} \hat{\mathbf{x}}_A^B & \hat{\mathbf{y}}_A^B & \hat{\mathbf{z}}_A^B \end{bmatrix}^T = \begin{bmatrix} (\hat{\mathbf{x}}_A^B)^T \\ (\hat{\mathbf{y}}_A^B)^T \\ (\hat{\mathbf{z}}_A^B)^T \end{bmatrix} \tag{4.10}$$

They will be used in frame transformations (Section 4.1.3), in the pinhole camera model (Section 4.2.1) and as an intermediate variable when obtaining the camera quaternion from the data.

Quaternion

Quaternions are the required input for SensorDTM to orient the camera for image rendering. It is also the most reliable method through which the same can be achieved in Blender. They are comprised of a real part, q_0, and three imaginary parts, $q_i, i \in \{1,2,3\}$. In SensorDTM, the quaternion is defined as $\mathbf{q}' = \begin{bmatrix} q_1 & q_2 & q_3 & q_0 \end{bmatrix}^T$. In the remaining applications, the quaternion is given as $\mathbf{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$.

Transformation from Rotation Matrices to Quaternions

To transform from rotation cosine matrices to their corresponding quaternions, the following method, taken from Diebel [2006], was used:

- Calculate possible quocients
  - $w_1 = \sqrt{max(1 + R_{1,1} + R_{2,2} + R_{3,3}, 0)}$
  - $w_2 = \sqrt{max(1 + R_{1,1} - R_{2,2} - R_{3,3}, 0)}$
  - $w_3 = \sqrt{max(1 - R_{1,1} + R_{2,2} - R_{3,3}, 0)}$
  - $w_4 = \sqrt{max(1 - R_{1,1} - R_{2,2} + R_{3,3}, 0)}$
- Pick maximum quocient, w_{max}
- Calculate quaternion using the formula associated to the maximum quocient
  - $w_{max} = w_1$: $\mathbf{q} = \frac{1}{2} \begin{bmatrix} w_1 & \frac{R_{2,3} - R_{3,2}}{w_1} & \frac{R_{3,1} - R_{1,3}}{w_1} & \frac{R_{1,2} - R_{2,1}}{w_1} \end{bmatrix}^T$

- $w_{max} = w_2$: $\mathbf{q} = \frac{1}{2}\begin{bmatrix} \frac{R_{2,3}-R_{3,2}}{w_2} & w_2 & \frac{R_{1,2}+R_{2,1}}{w_2} & \frac{R_{3,1}+R_{1,3}}{w_2} \end{bmatrix}^T$
- $w_{max} = w_3$: $\mathbf{q} = \frac{1}{2}\begin{bmatrix} \frac{R_{3,1}-R_{1,3}}{w_3} & \frac{R_{1,2}+R_{2,1}}{w_3} & w_3 & \frac{R_{2,3}+R_{3,2}}{w_3} \end{bmatrix}^T$
- $w_{max} = w_4$: $\mathbf{q} = \frac{1}{2}\begin{bmatrix} \frac{R_{1,2}-R_{2,1}}{w_4} & \frac{R_{3,1}+R_{1,3}}{w_4} & \frac{R_{2,3}+R_{3,2}}{w_4} & w_4 \end{bmatrix}^T$

where $w_i, i \in \{1,2,3,4\}$ is the quocient used for the quaternion calculation in option i; $max(x,y)$ is the maximum value of a given list; w_{max} is the maximum of the quocient values; and $R_{i,j}$ is the entry at line i and column j of the rotation matrix $\mathbf{R}$. Using the maximum quocient is protective against numerical errors from divisions with small numbers (which was observed before this conditional calculation was implemented).

Transformation from Quaternions to Rotation Matrices

To obtain the rotation matrix associated with a quaternion the following equation is used [Sidi, 2006]:

$$\mathbf{R} = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_1q_2) \\ 2(q_1q_2-q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} \tag{4.11}$$

4.1.3. Frame Transformations

This section provides the frame transformations used, namely from the 3D model frame, $\mathscr{F}_M$, to the camera frame, $\mathscr{F}_C$; from $\mathscr{F}_C$ to the image frame, $\mathscr{F}_i$, and vice-versa; from $\mathscr{F}_M$ to $\mathscr{F}_i$ and vice-versa; and from $\mathscr{F}_i$ to $\mathscr{F}_z$.

Transformation from 3D Model Frame ($\mathscr{F}_M$) to Camera Frame ($\mathscr{F}_C$)

This transformation is achieved via a translation by $-\mathbf{T}_C^M$, followed by a rotation by $\mathbf{R}_M^C$, as expressed by:

$$\mathbf{p}^C = \mathbf{R}_M^C\left(\mathbf{p}^M - \mathbf{T}_C^M\right) \tag{4.12}$$

where $\mathbf{T}_C^M$ is the position vector of the camera in $\mathscr{F}_M$. This transformation can also be done using the extrinsic camera matrix, $\mathbf{E}$, as explained in Section 4.2.1

$$\mathbf{p}^C = \mathbf{E}\mathbf{p}^M \tag{4.13}$$

Transformation between Camera Frame ($\mathscr{F}_C$) and Image Frame ($\mathscr{F}_i$)

The transformation from the camera frame, $\mathscr{F}_C$, and the image frame, $\mathscr{F}_i$, is made using the intrinsic camera matrix, $\mathbf{K}$, as explained in Subsection 4.2.1

$$\mathbf{p}^i = \mathbf{K}\mathbf{p}^C \tag{4.14}$$

The inverse transformation is obtained with the inverse of the intrisic matrix by

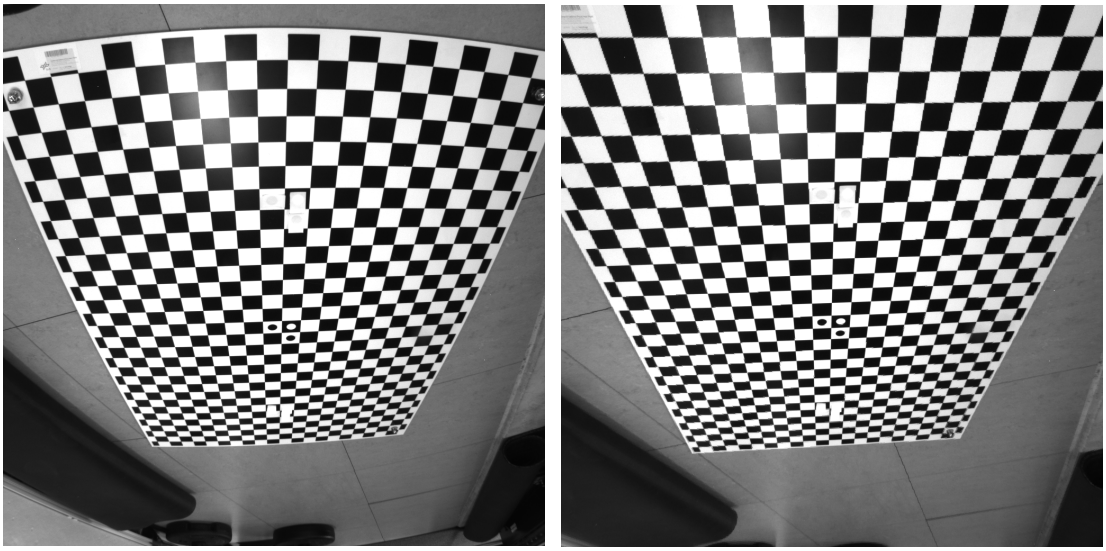
$$\mathbf{p}^C = \mathbf{K}^{-1}\mathbf{p}^i \tag{4.15}$$

Transformation between 3D Model Frame ($\mathscr{F}_M$) and Image Frame ($\mathscr{F}_i$)

This transformation from $\mathscr{F}_M$ to $\mathscr{F}_i$ is made by first transforming from $\mathscr{F}_M$ to $\mathscr{F}_C$, followed by a transformation to $\mathscr{F}_i$. Alternatively, as explained in Section 4.2.1, it can also be made using the camera calibration matrix, $\mathbf{C}$.

$$\mathbf{p}^i = \mathbf{C}\mathbf{p}^M \tag{4.16}$$

The inverse transformation requires depth data. The process is explained in Section 4.3.



(a) Original image (as taken by camera)

(b) Undistorted image (corrected for camera distortion)

Figure 4.4: Comparison between original and undistorted images of a checkerboard

Transformation from Image Frame ($\mathscr{F}_i$) to Depth Data Frame ($\mathscr{F}_z$)

To find the depth data for a given image pixel, the coordinates in which the corresponding data is stored (in $\mathscr{F}_z$) are obtained via

$$x^z = round\left(x^i - 0.5\right) \tag{4.17}$$

and similarly for y^z, where $round(x)$ is the closest integer to x.

4.2. Camera Model

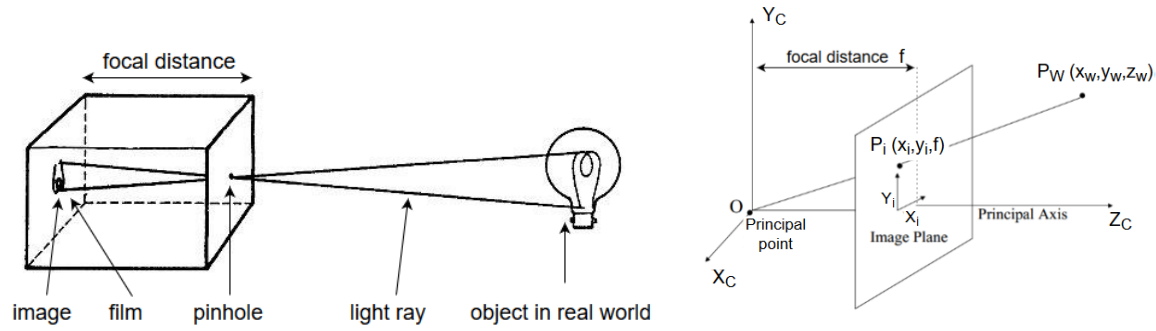
The proposed software makes use of image rendering and processing, for which a camera model is required. Additionally, this model is also necessary to derive the 3D world coordinates of a point from its 2D image coordinates and depth data.

In practice, a camera is a mechanism that projects 3D (real world) points onto a 2D surface (image plane). A camera can be constructed as a dark compartment with a small opening (aperture) through which light can enter. The light is then focused through a (series of) lens(es) onto the image plane. The amount of light that hits a given sensor element determines the value of the corresponding pixel in the image. Changing the exposure time (time period in which the sensors acquire information for one image) will affect the brightness and contrast of an image.

Models representative of cameras range from highly realistic – such as the one described by Kolb et al. [1995], where the light rays through the various lenses are modelled – to very simple ones – such as the orthographic projection, where an image point is the intersection between the film and a line perpendicular to it that passes through the corresponding world point.

The rendering softwares that will be used represent the camera with the pinhole camera model. This model is also adequate for the processing required, namely finding a correspondence between a given pixel in the image captured by the camera (2D) and the respective real world point (3D). It accounts for the camera characteristics (*e.g.*, focal length) and camera pose, thus providing a direct correspondence between the points in the real world (*e.g.*, in metres) and the points in the image coordinate system (in pixels). Section 4.2.1 provides a description of this model.

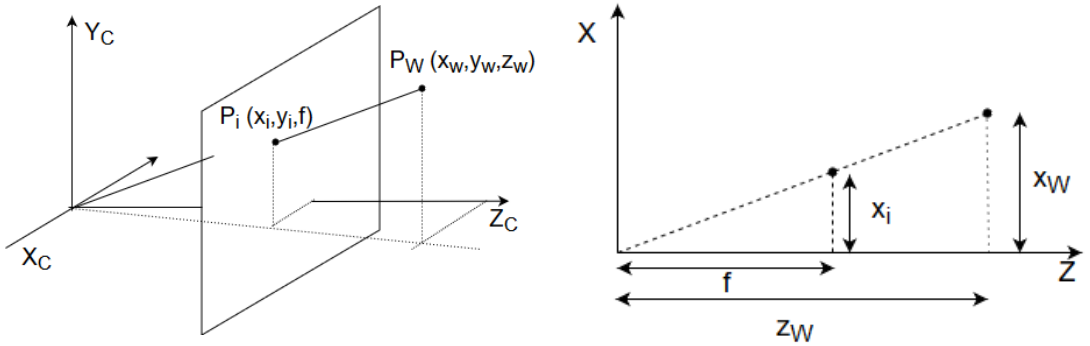
It is also important to model some degree of distortion, since this effect can displace image pixels considerably and one pixel in the image can correspond to hundreds of metres in the real world. Distortion refers to any effect that makes straight lines in the real world be represented as curved lines in the image. This is



(a) Simple theoretical construction of a pinhole camera

(b) Equivalent representation

Figure 4.5: Pinhole model representations [1]



(a) Representation of projection to image plane

(b) Equivalent triangles for the x-direction

Figure 4.6: Perspective projection

visible in Figure 4.4, where an image of a checkerboard and its undistorted counterpart are compared. A brief introduction to the different kinds of distortion is given in Section 4.2.2.

4.2.1. Pinhole Camera Model

In its simplest form, the pinhole camera is a dark box with an (infinitesimal) hole (principal point) in one face and a film in the inner side of the parallel face, as shown in Figure 4.5a.

As represented in Figure 4.5a, for each point in the real world only one of the reflected light rays will enter the camera. The ray hits the film imprinting the corresponding tonal value in the image plane.

In this representation, the resulting image is upside down (a vertical mirroring of reality). To avoid this, we can represent the model as in Figure 4.5b, where the image plane is moved to the front of the pinhole, keeping the same distance (focal length) to it.

If we take the pinhole as the origin of the coordinate system, a point $\mathbf{P}_W$ in the real world will correspond to point $\mathbf{P}_i$ in the image plane. A visual representation of this projection is shown in Figure 4.6a. For a distance of f between the pinhole and the image plane (focal length), the $\mathbf{P}_i$ coordinates can be obtained from the $\mathbf{P}_W$ coordinates, through Equation (4.19), using equivalent triangles as shown in Figure 4.6b and

$$\begin{aligned} \frac{x_i}{f} &= \frac{x_W}{z_W} \\ x_i &= f\frac{x_W}{z_W} \end{aligned} \tag{4.18}$$

[1] Adapted from /http://practicalphysics.org/images/From%20pinhole%20camera%20to%20lens%20camera_1217.jpg, and /http://www.ics.uci.edu/~majumder/VC/classes/cameracalib.pdf, respectively, last access: 18/04/2018

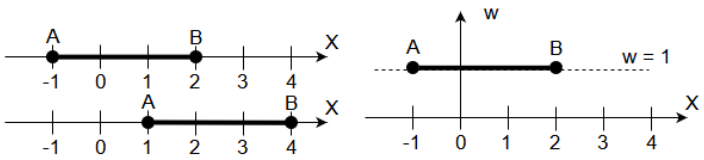


Figure 4.7: Representations of rod in the 1D world (left) and in the augmented 2D homogeneous coordinates (right).

$$
\begin{aligned}
x_i &= \frac{f x_W}{z_W} \\
x_i &= \frac{f y_W}{z_W} \\
z_i &= f
\end{aligned}
\tag{4.19}
$$

Transforming the projected point, $\mathbf{P}_i$, from the camera frame, $\mathcal{F}_C$, to the image frame, $\mathcal{F}_i$, is done by taking the first two coordinates – x_i^C and y_i^C – as given by

$$
\mathbf{P}_i^i = \begin{pmatrix} x_i^i \\ y_i^i \end{pmatrix} = \begin{pmatrix} x_i^C \\ y_i^C \end{pmatrix} = \begin{bmatrix} \frac{f}{z_W^C} & 0 & 0 \\ 0 & \frac{f}{z_W^C} & 0 \end{bmatrix} \begin{pmatrix} x_W^C \\ y_W^C \\ z_W^C \end{pmatrix} = \begin{bmatrix} \frac{f}{z_W^C} & 0 & 0 \\ 0 & \frac{f}{z_W^C} & 0 \end{bmatrix} \mathbf{P}_W^C \tag{4.20}
$$

This method of transforming coordinates is not ideal, since the transformation matrix depends on one of the coordinates of the real world point. Thus, it is different for every non-coplanar point. Since, for the application at hand, a unique matrix greatly simplifies the problem, the concept of homogeneous coordinates is introduced next. This concept makes it possible for perspective projections (as the one used for the pinhole camera model) and translations to be written as simple unique matrices independent of the real world coordinates of a point.

Homogeneous Coordinates

1D Definition

Take a 1D world, where a rigid rod of length 3 lies. Points A and B correspond to the ends of this rod, as shown in Figure 4.7 (top left). At the first instance, $x_A = -1$ and $x_B = 2$. If the rod is moved by two units to the right, the translation of the two ends can be given by

$$
\begin{aligned}
x_{A'} &= 1 = [-1] x_A \\
x_{B'} &= 4 = [2] x_B
\end{aligned}
\tag{4.21}
$$

where x_P and $x_{P'}$ refer to the coordinates of point P before and after the translation, respectively. In other words, the two points would require different transformation matrices: $[-1]$ for point A and $[2]$ for point B.

To avoid this, a new fictitious coordinate can be introduced, such that the 1D world is contained in the line $w = 1$, as seen in Figure 4.7 (right). In other words, the rod can only exist in this $w = 1$ line. With this new set-up, the same translation could be achieved with one single transformation matrix $\mathbf{T}$ by

$$
\begin{aligned}
\mathbf{T} \begin{pmatrix} x_A \\ w_A \end{pmatrix} &= \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \times (-1) + 2 \times 1 \\ 0 \times (-1) + 1 \times 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} x_{A'} \\ w_{A'} \end{pmatrix} \\
\mathbf{T} \begin{pmatrix} x_B \\ w_B \end{pmatrix} &= \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \times 2 + 2 \times 1 \\ 0 \times (2) + 1 \times 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \end{pmatrix} = \begin{pmatrix} x_{B'} \\ w_{B'} \end{pmatrix}
\end{aligned}
\tag{4.22}
$$

The new set of coordinates - (x, w) - are the homogeneous coordinates.

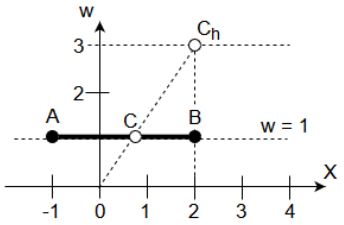


Figure 4.8: Physical meaning of points with coordinate $w \neq 1$.

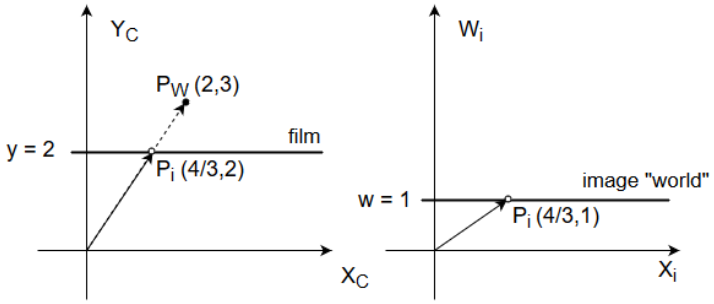


Figure 4.9: Perspective projection for $f = 2$ in the camera frame (left) and result in homogeneous coordinates (right).

Physical Meaning of Points Outside the $w = 1$ Line

Now, let C be a point in the rod for which the given homogeneous coordinates are $(2,3)$. These coordinates seem to contradict the fact that the entire world in which the rod can exist in is contained in $w = 1$. To correct this incongruence, the vector defining point C needs to be scaled such that $w = 1$, as shown in Figure 4.8.

The result is that point C is actually at $x = 2/3$. In other words, point C – $x = 2/3$ – can be expressed in homogeneous coordinates through any vector of the form $k(2/3, 1)$, where $k \in \mathbb{R}$. In homogeneous coordinates, $(2/3, 1)$, $(2,3)$ and $(-4,-6)$ all represent the same point (hence, the name).

Perspective Projection from a 2D World Onto a 1D Image

Taking a closer look at Figure 4.8, point C can be thought as the perspective projection of a real world 2D point C_h, using a camera with $f = 1$. So for $f = 1$, the homogeneous image coordinates of an image point are the same as the camera frame coordinates of the corresponding real world point.

For $f \neq 1$ an additional transformation is required. Figure 4.9 shows the real world point P_W projected in a film located in the line $y = 2$, meaning $f = 2$. The resulting point P_i is represented in the camera (left) and homogeneous (right) coordinate systems.

Through observation, this perspective projection can be thought of as scalling the y-axis of the real world coordinate system to fit the w-axis of the homogeneous coordinate system. In other words, the homogeneous coordinates of point P_i are

$$x_i^i = x_W^C = 2$$
$$w_i^i = \frac{y_W^C}{f} = \frac{3}{2}$$

or equivalently

$$x_i^i = f\,x_W^C = 2 \times 2 = 4$$
$$w_i^i = y_W^C = 3$$

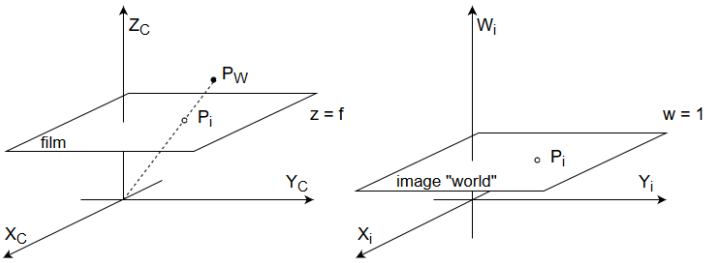


Figure 4.10: Use of homogeneous coordinates to represent points in an image plane. Left - real world 3D coordinates; right - homogeneous coordinates in the 2D image plane.

Extending for 2D and 3D Worlds

The concept of homogeneous coordinates can be extended for 2D and 3D spaces [2]. To do so, it is sufficient to introduce one extra coordinate, w, and place the original 2D or 3D world at the $w = 1$ plane or hyper-plane, respectively.

The 2D case (represented in Figure 4.10) is helpful to describe points in the image plane. The 2D image points are now contained in the plane $w = 1$ in the homogeneous coordinate system. The perspective projection transformation is done in a way similar to the 1D case. The resulting homogeneous coordinates for the projected point are $\begin{pmatrix} fx_p & fy_p & z_p \end{pmatrix}^T$.

In the 3D case for the homogeneous coordinates, the real world points are contained in the hyperplane $w = 1$. These coordinates can be used, for example, to facilitate translations.

Calibration Matrix In the ideal case, we can get the projected point $\mathbf{P}_i^i$ from Equation (4.23) [3],

$$\mathbf{P}_i^i = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{P}_W^C = \mathbf{K}\mathbf{P}_W^C \tag{4.23}$$

where $\mathbf{K}$ is the intrinsic parameter matrix; $\mathbf{P}_i^i$ is given in homogeneous coordinates in $\mathscr{F}_i$; and $\mathbf{P}_W^C$ in 3D Cartesian coordinates in $\mathscr{F}_C$.

However, this would require the origin of the image frame to be at the projection of the principal point, which is not the case (see Subsection **??**). Aditionally, real cameras may have non-orthogonal and non-square sensors; and the image is represented in pixels.

Figure 4.11 shows the two different coordinate systems, where the ideal frame is $\mathscr{F}_i'$. C_0 is the projection of the principal point in the image plane and (u_0, v_0) are its coordinates in $\mathscr{F}_i$ given in pixels (in the example (4/3,3)). θ is the angle between the X_i- and the Y_i-axis, which, most commonly, is approximately 90°. The parameters m_u and m_v are the dimensions of a pixel in real world units (*e.g.*, mm). For square pixels m_u and m_v have the same value.

Using these variables, in the general case, $\mathbf{K}$ is given by Equation (4.24), from [Forsyth and Ponce, 2003, p. 131],

$$\mathbf{K} = \begin{bmatrix} f_u & -f_u \cot\theta & u_0 \\ 0 & \frac{f_v}{\sin\theta} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.24}$$

where f_u and f_v are the magnifications given by

$$\begin{aligned} f_u &= \frac{f}{m_u} \\ f_v &= \frac{f}{m_v} \end{aligned} \tag{4.25}$$

[2] Note that despite involving a 2D world, the previous case still only applied homogeneous coordinates to the 1D image film

[3] /http://www.ics.uci.edu/~majumder/VC/classes/cameracalib.pdf, last access: 18/04/2018

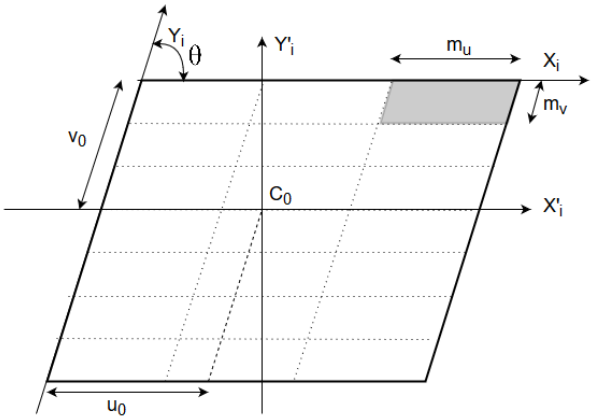


Figure 4.11: Ideal and general image coordinate systems.

Since **K** depends only on the characteristics of the camera, its parameters correspond to the intrinsic calibration of the camera (discussed in Section[camera calibration]).

In adition, the camera frame is not the same 3D model frame (in which the world points will be described). In other words, before projecting the world points onto the image, they have to be transformed from $\mathcal{F}_M$ to $\mathcal{F}_C$.

As discussed in Subsection 4.2.1, translations can be simplified by using homogeneous coordinates. Namely, given a point $\mathbf{P}_W$, it can be translated by vector **T** through

$$\mathbf{P}'_W = [\mathbf{I}|\mathbf{T}]\,\mathbf{P}_W \tag{4.26}$$

where $\mathbf{T} = \begin{pmatrix} T_x & T_y & T_z \end{pmatrix}^T$ is the translation vector and **I** a 3×3 identity matrix.

Finally, the result can be pre-multiplied by the rotation matrix $\mathbf{R}^C_M$. The final matrix to align the camera is $\mathbf{E} = \mathbf{R}[\mathbf{I}|\mathbf{T}]$, named extrinsic parameter matrix, since it only depends on the external conditions.

The complete projection is given by

$$\mathbf{P}^i_i = \mathbf{KEP}^M_W = \mathbf{CP}^M_W \tag{4.27}$$

where $\mathbf{P}^i_i$ and $\mathbf{P}^C_W$ are both in homogeneous coordinates, 3D and 4D respectively; and **C** is a 3×4 matrix called complete camera calibration matrix, with 11 unknown parameters, five from the intrinsic calibration (f_u, f_v, θ, u_0 and v_0), three from the translation (T_x, T_y and T_z) and the three rotation angles.

4.2.2. Distortion

Geometrical distortion (as previously mentioned) is any effect that causes straight lines in the real world to appear curved in the image. An example of distortion was shown in Figure 4.4.

These distortions are caused by lens imperfections. Since the DLR CalLab tool is the one to be used for camera calibration (see Section 5.6), the distortion models used in the program are the ones discussed below. These are similar to the radial, decentring and thin prism distortions modelled by Weng et al. [1992].

The distortion is a displacement between the ideal location of a point in the image (if the camera was perfect) with respect to the actual position of that pixel. The degree of displacement depends on the position of the ideal point in the image. The distorted image coordinates can be written as a function of the ideal ones as

$$\begin{aligned} u' &= u + \delta_u(u, v) \\ v' &= v + \delta_v(u, v) \end{aligned} \tag{4.28}$$

where (u, v) are the ideal point's coordinates (in the image frame where the origin is the principal point); (u', v') are its distorted coordinates; and δ_u and δ_v are, respectively, the displacements along the u and v coordinates.

In Subsection 4.2.2, a brief description of the origin of the three different types of distortions considered by Weng et al. [1992] is given. Subsection 4.2.2 explains how distortion is handled by OpenCV, which will be used to undistort images and handle distortion during pose determination.

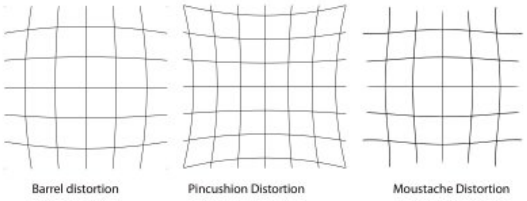


Figure 4.12: Radial distortions [4].

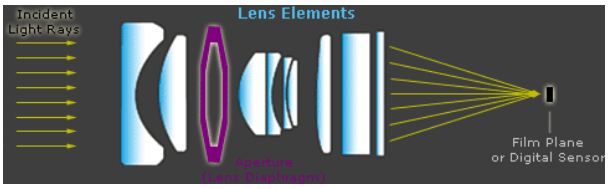


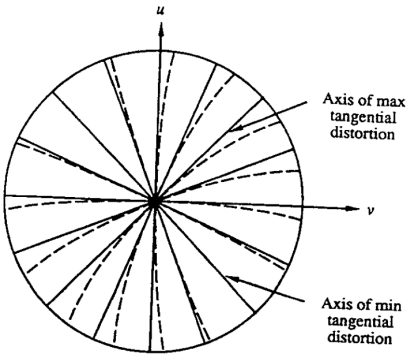
Figure 4.13: Lens elements. [5]

Geometric Distortion

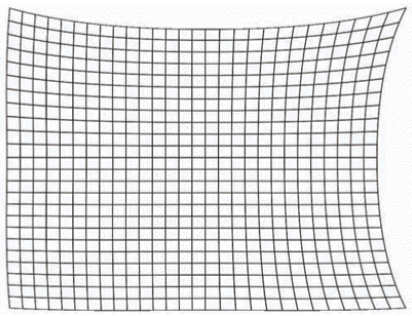
Radial Distortion is caused by imperfections in the lens shape and results only in radial displacement (hence the name). It is modelled as a function of the radial distance to the image's principal point. Two opposing extreme cases of radial distortion are barrel and pincushion distortions, represented in Figure 4.12 (left and center, respectively), which shows the effect of these forms of distortion on a grid. Generaly, a subtle combination of the two shapes is observed in an image (right image of Figure 4.12).

Decentring Distortion is caused by improper lens and camera assembly and results from the fact that different lens elements are not perfectly aligned, meaning their optical centres are not collinear. An example of the positioning of some lens elements is shown in Figure 4.13. This effect leads both to radial and tangential displacements. Figures 4.14a and 4.14b show the effect of tangential distortion in an image.

Thin Prism Distortion is a result of imperfections in the lens design and manufacturing, as well as assembly errors. This distortion also leads to radial and tangential distortions.



(a) Tangential distortion. Solid lines: no distortion. Dashed lines: with tangential distortion. [Weng et al., 1992].



(b) Effect of tangential distortion on a grid. [Ribbens et al., 2013]

Figure 4.14: Tangential distortion effects.

[4] /https://www.lensrentals.com/blog/media/2010/10/blog13.jpg, last access: 09/01/2018

[5] /http://curta.dlinkddns.com/html_photography/Understanding_Camera_Lenses/tut_lensflare_elements.png, last access: 09/01/2018

Camera Distortion in OpenCV For the applications, camera distortion will be handled in OpenCV, which can take five distortion parameters (three for radial distortion and two for tangential distortion) [6]. The corrections are made using

$$\begin{aligned} x_{corrected} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{corrected} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{4.29}$$

for the radial distortion and

$$\begin{aligned} x_{corrected} &= x + (2p_1 xy + p_2(r^2 + 2x^2)) \\ y_{corrected} &= y + (p_1(r^2 + 2y^2) + p_2 xy) \end{aligned} \tag{4.30}$$

for the tangential distortion, where r is the radial distance from an image point to the principal point. This is the form in which the distortion parameters for the reference cameras are provided both for CE-3 and by CalLab (see Section 5.6).

4.3. World Coordinates from Pixel Coordinates

The database to be prepared offline will include the coordinates in $\mathscr{F}_M$ of the features (see Chapter 3) found in the rendered images. The detection provides the coordinates of the features in $\mathscr{F}_i$. The transformation from $\mathscr{F}_i$ to $\mathscr{F}_M$ can be achieved if the individual $\mathbf{K}$, $\mathbf{R}_M^C$ and $\mathbf{T}_C^M$ matrices are known and some depth data for the image points are given.

When using Blender, the depth information available is the z-depth: distance from the camera to the world point corresponding to each image pixel; for SensorDTM, it is the z-value: the z_C-coordinate for the same point.

The method used is

- Define direction vector, $\mathbf{d}^i = \begin{pmatrix} u_i & v_i & 1 \end{pmatrix}^T$
- Transform into the ideal image frame $\mathbf{d}^C = \mathbf{K}^{-1}\mathbf{d}^i$
- Transform into the 3D model frame $\mathbf{d}^M = (\mathbf{R}_M^C)^{-1}\mathbf{d}^C$
- Scale direction vector
  - If z-depth data (η): $\mathbf{d}_S^M = \mathbf{d}^M \frac{\eta}{\|\mathbf{d}^M\|}$
  - If z-value data (η): $\mathbf{d}_S^M = \mathbf{d}^M \frac{\eta}{\mathbf{d}_z^M}$
- Perform frame translation: $\mathbf{p}_W^M = \mathbf{d}_S^M + \mathbf{T}_C^M$

where $\mathbf{d}^X$ is the direction vector pointing to the world point in $\mathscr{F}_X$; u_i and v_i are the feature's image coordinates; $\mathbf{d}_S^M$ is the scaled direction vector from the origin of $\mathscr{F}_C$ to the world point in $\mathscr{F}_M$; and $\mathbf{p}_W^M$ is the world point in $\mathscr{F}_M$.

The steps of the algorithm are represented in Figure 4.15 (to be read, starting at the top, from left to right). The feature is represented by a circle, and the corresponding world point by a dot. The first image shows the feature in the $\mathscr{F}_i$ frame. The $\mathbf{d}^i$ vector is defined based on the feature's pixel coordinates. Using the inverse of the camera matrix, $\mathbf{K}$, the vector is transformed into the $\mathscr{F}_C$ frame, as shown in the second image (top middle). The same vector is then described in the $\mathscr{F}_M$ frame (top right image), which is achieved with the adequate rotation matrix.

Using the depth data, the vector is scaled so that it points from the camera's principal point to the world point associated to the feature (bottom left). Finally, adding the position vector of the camera, $\mathbf{T}_C^M$, the position of the world point is determined in the $\mathscr{F}_M$ frame (bottom right).

4.4. Pose Determination

The goal of the proposed software is to retrieve the camera pose used in capturing a query (online) image, through the matches found between the features detected in it and the ones in the database.

The features are found through one of the algorithms described in Sections 3.2 and 3.3. The database features are used, along with depth data, to get their corresponding world point coordinates (3D coordinates, see Section 4.3), which are saved as the data feature points. The corresponding descriptors are then used to

[6] /http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html, last access: 18/04/2018

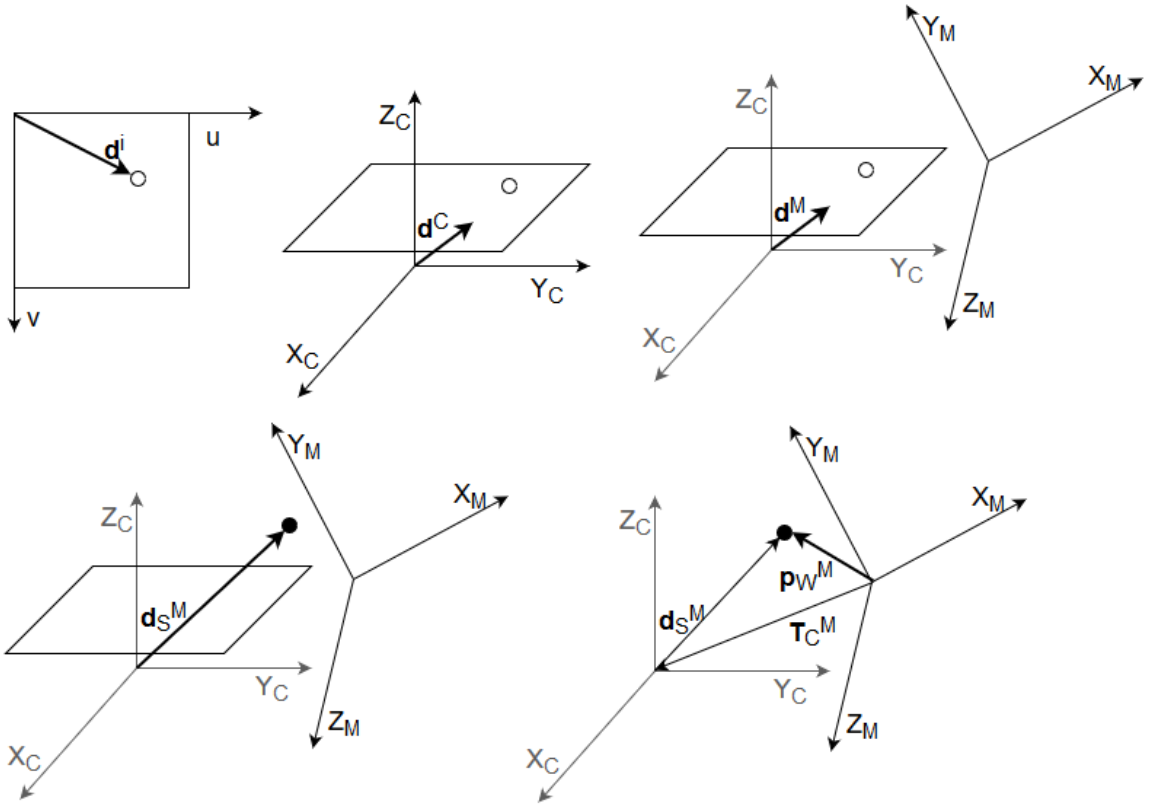


Figure 4.15: Derivation of world coordinates from feature coordiantes

build a FLANN index (Subsection 3.6.2). The NN solutions for the descriptors of the query features are found using this index and down-selected as discussed in Section 3.7.

In the end of this process, there are N matches between the image coordinates of the query features (2D) and their corresponding world point coordinates (3D) in $\mathcal{F}_M$. Some of these matches can be incorrect. However, it was argued in Section 3.6 that the down-selection performed is able to significantly reduce their percentage.

With this correspondence of 2D image coordinates to 3D world coordinates, it is possible to obtain the camera pose, by solving a Perspective n-Point (PnP) problem, which corresponds to finding the extrinsic parameter matrix that transforms the world points into the image points, provided the intrinsic parameter matrix and the distortion parameters.

The algorithm chosen to solve this problem was the EPnP algorithm available in OpenCV (Subsection 4.4.1). To account for the possible incorrect matches, the Random Sample Consensus (RANSAC) version of this algorithm was used (Subsection 4.4.2).

4.4.1. Effective Perspective n-Point (EPnP)

The EPnP algorithm was introduced by Lepetit et al. [2008] to which the reader is referred to for the mathematical details. It is an algebraic algorithm, meaning the solution is found by solving a set of linear equations. After finding the initial solution, the algorithm also includes a fast optimisation that improves the accuracy of the final solution, without a significant increase of the computation time.

One important characteristic of this algorithm is that it begins by computing the orientation of the camera. This result is then used in calculating the position. As a result, errors in the position are correlated to errors in the orientation [Trigo et al., 2018]. This has been verified during this research (Subsection 7.2.4).

The method starts by defining all the world points (reference points), $\mathbf{p}_i$, considered as linear combinations of four virtual points (control points), $\mathbf{c}_j$. The combination parameters (homogeneous barycentric co-

ordinates) – α_{ij} – used to describe $\mathbf{p}_i$ as a linear combination of $\mathbf{c}_j$, are unique and easily estimated. They are calculated using the reference points as they are given in $\mathscr{F}_M$, resulting in the relation

$$\mathbf{p}_i^M = \sum_{j=1}^{4} \alpha_{ij} \mathbf{c}_j^M, \sum_{j=1}^{4} \alpha_{ij} = 1 \tag{4.31}$$

This relation also holds true if using the points in $\mathscr{F}_C$, as in

$$\mathbf{p}_i^C = \sum_{j=1}^{4} \alpha_{ij} \mathbf{c}_j^C \tag{4.32}$$

If $\mathbf{p}_i$ are known in $\mathscr{F}_C$, then $\mathbf{c}_j^C$ can be calculated. The correspondence between $\mathbf{c}_j^M$ and $\mathbf{c}_j^C$ provides a unique solution for the transformation between the two frames.

The reference points in $\mathscr{F}_C$, $\mathbf{p}_i^C$, can be determined from $\mathbf{p}_i^i$ (image points matched to the world reference points), the intrinsic parameter matrix, $\mathbf{K}$, and an unknown scalar perspective parameter (depth data) for each reference point (after correcting for distortion). Through mathematical manipulation, these unknown parameters are eliminated from the equations.

The final formulation contains $2n$ equations, where n is the number of reference points, with 12 unknown variables, corresponding to the coordinates of the four control points in $\mathscr{F}_C$. The solution is calculated from the null eigenvectors of $\mathbf{M}^T\mathbf{M}$, where $\mathbf{M}$ is the matrix representing the aforementioned equations.

To increase the accuracy, the solution is further optimised by minimising the reprojection error. Using the solution obtained, $\mathbf{K}$ and the distortion parameters, the $\mathbf{p}_i^M$ are projected onto the image plane. The distance between the projected points and $\mathbf{p}_i^i$ is used as the cost function. In practice, the optimisation requires less than 10 iterations.

The outputs are the translation and rotation vectors for the transformation from $\mathscr{F}_M$ to $\mathscr{F}_C$. To get the inverse transformation, first the Rodrigues rotation vector is transformed into a direction cosine matrix (namely $\mathbf{R}_M^C$) via the OpenCV *Rodrigues* function. The full transformation matrix is defined by

$$\mathbf{A}_M^C = \begin{bmatrix} \mathbf{R}_M^C & \mathbf{T}_M^C \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{4.33}$$

where $\mathbf{T}_M^C$ is the translation vector as given by the solution; and $\mathbf{0}_{1\times 3}$ is a 1×3 null matrix.

The inverse of this matrix provides the transformation from $\mathscr{F}_C$ to $\mathscr{F}_M$ and thus the camera pose. In other words

$$\left(\mathbf{A}_M^C\right)^{-1} = \mathbf{A}_C^M = \mathbf{E} = \begin{bmatrix} \mathbf{R}_C^M & \mathbf{T}_C^M \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{4.34}$$

4.4.2. Random Sample Consensus (RANSAC)

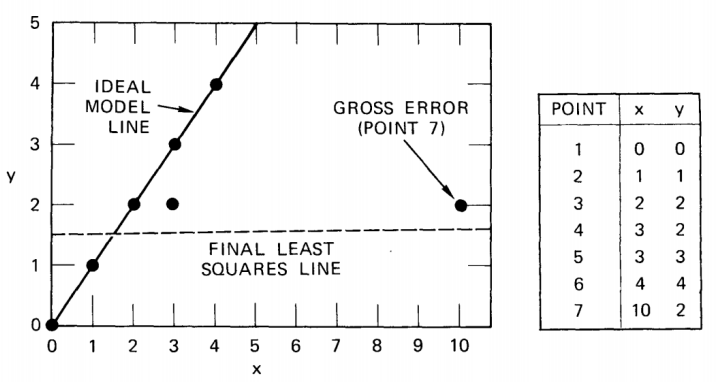
As mentioned in Chapter 3, the matching process does not guarantee a perfect correspondence between the data and query features. Thus, despite the down-selection performed (see Section 3.7), it is possible for outliers to be included in the reference points. EPnP by itself cannot handle outliers. To do so, the RANSAC scheme, introduced by Fischler and Bolles [1981], can be used.

Figure 4.16 shows how regular fitting methods can significantly deviate from the ideal model in the presence of outliers. RANSAC, on the other hand, can yield very good solutions independently of these points, since it is capable of ignoring them.

RANSAC begins by randomly selecting a small sub-set of the dataset and using it to find the best fitting parameters of a mathematical model. Each of the points in the dataset are checked to determine if they are inliers. Outliers are ignored. If a sufficient number of inliers exists, they are used to calculate the error associated with the model. This is repeated until a certain number of iterations is achieved or enough inliers are found.

Figure 4.17 provides a simple example on how RANSAC works. In this case, a straight line is being fit to a set of 2D points, represented by dots. To fit a straight line only two points are required. Thus, for each try, RANSAC selects two random points (represented by circles) and finds the line parameters associated (slope and the y-intercept). Then, the distance of each point to the line is calculated.

The threshold for this distance (represented by the dashed line) is used to discriminate between the inliers and outliers. For the first attempt (left), only the two points used to generate the line are within the acceptable range. All other points are considered outliers. Since the number of outliers is too high, this solution is discarded. In the second attempt, only one point is (correctly) recognised as outlier, so this solution is accepted. In some applications, the set of inliers obtained is used to derive a better model.



POINT	x	y
1	0	0
2	1	1
3	2	2
4	3	2
5	3	3
6	4	4
7	10	2

Figure 4.16: Comparison between line fit to data using Least Squares (LSQ) and the ideal model line, $y = x$ [Fischler and Bolles, 1981]

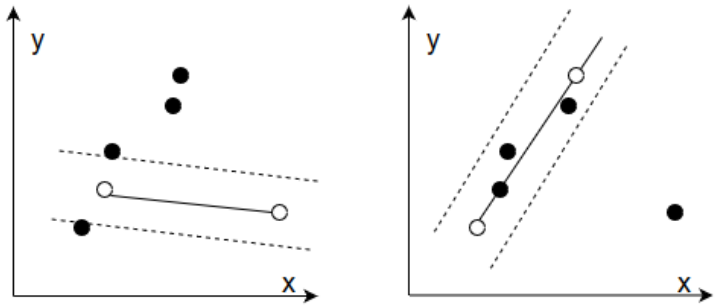


Figure 4.17: Example of line fitting with RANSAC

For a given dataset; a mathematical model; the number of point to fit the model, n; the maximum number of iterations allowed, k; a threshold to determine whether a data point is an inlier or not, t; and minimum number of inliers required d, the method used is as follows

- Randomly select n data points
- Find the parameters that best fit the model to selected points
- Find all the dataset points for which the modelling error is smaller than t; and save them as inliers
- If the number of inliers is greater than d: save the parameters and stop
- Otherwise: repeat

For the EPnP algorithm in OpenCV, $n = 5$.

Since it relies on random point selection, RANSAC does not guarantee a solution, and in case one is obtained, it may not be the optimal one. This is mostly an issue for large percentages of outliers (above 50%), which are not expected as a consequence of the pre-selection performed (Section 3.7).

5 | Testbed for Robotic Optical Navigation (TRON)

To properly analyse the navigation accuracy of the algorithm developed, real measurements are preferred over simulations, particularly due to the use of rendered (simulated) images to build the database: given that both the data and online images would be generated through the same method, this could skew the simulated results. On one hand, these measurements should include real images representative of the lunar surface, to validate the image processing methods used (discussed in Chapter 3) in the context of interest. On the other hand, they should also include ground-truth for the position of the camera in the 3D world (discussed in Chapter 4) with higher precision than the one required for the navigation solution, for comparison with the navigation solution.

Part of analysing the theoretical potential of the software, as mentioned in Section 1.2, is the evaluation of the navigation solution results of using ideal data (DEMs of high resolution and accuracy, real trajectory and illumination conditions used). To do so, the data was acquired in TRON. This laboratory was created to provide reliable simulation data for the ATON project. As such, it has been equiped with simulation mechanisms, models and environment that can be used for lunar landing simulations.

Section 5.1 provides a description of the laboratory and the characteristics that make it suitable to simulate lunar landing camera images. The measurement system in Section 5.2 and the mechanism and methods used to place the camera in the desired positions are described in Section 5.3. The terrain model used is characterised in Section 5.4, followed by an explanation of the reference frames involved (Section 5.5). Finally, camera calibration is discussed in Section 5.6.

5.1. Testing Environment

Figure 5.1 represents the general layout of TRON. This facility is a 15 m long, 5.1 m wide and 3 m tall room divided into two sections: the operators section and the simulations section. This division has two purposes: first, to avoid interferences caused by external light sources; second, to provide protection from (laser) radiation and the movement of the robot (Section 5.3). The separation is made by a wall, which provides visual access through a window [Krüger et al., 2014].

In the operation section, the operator can use the host PC to run scripts for simulation hardware control, to capture images, make measurements (Section 5.2) and manage the data. Apart from the host PC, this section contains

- the real time simulation system dSPACE
- the manual controls for the robot and gantry

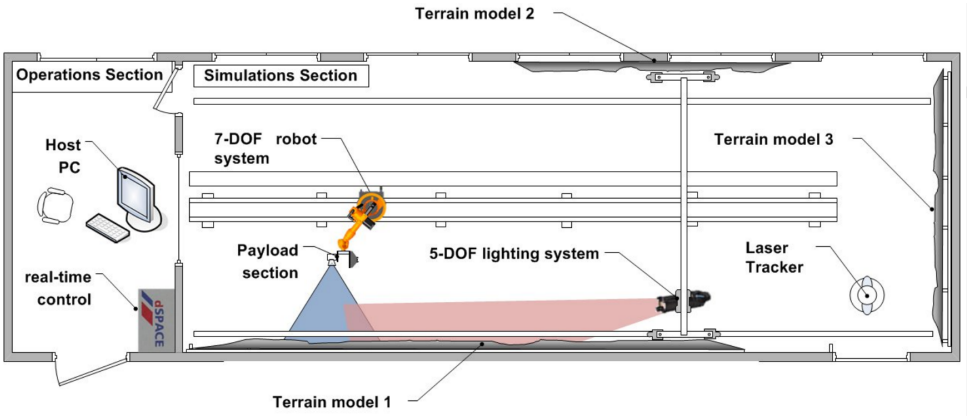


Figure 5.1: Layout of TRON [Krüger et al., 2014]

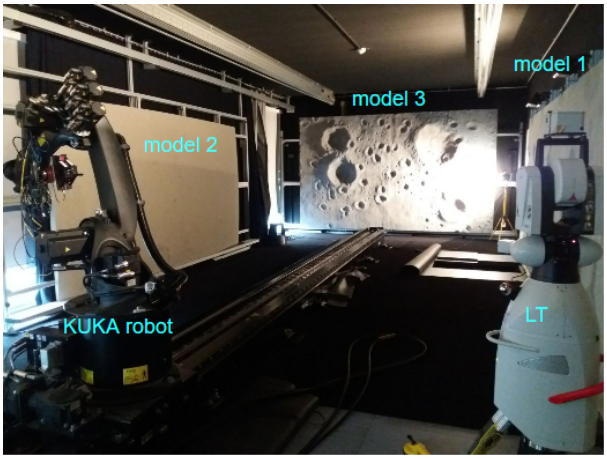


Figure 5.2: Simulation section of TRON

- the safety-system interfaces

Among the availaboratoryle scripts to control the simulation, one allows to run a predefined trajectory and synchronises the camera images with the measurements taken.

The simulation section, shown in Figure 5.2, includes a robot (Section 5.3) mounted on a rail running along the length of the room; a laser tracker (Section 5.2); three terrain models representative of the lunar surface; a computer to operate the laser-scanner; and a lighting system.

The lighting system consists of a lamp mounted on a gantry with five Degrees of Freedom (DoF). The lamp itself has a colour temperature of 6000 K and uniform lighting (making it representative of the Sun). It can be moved in any direction and rotated about two axis.

To better simulate the lighting conditions in the laboratory, it includes a black-out system consisting of movable curtains; and an anti-reflection system achieved through the black colour of all the room's surfaces, which minimises the secondary light sources.

5.2. Laser Tracker

The position and orientation measurements in the laboratory are performed using the laser tracker (AT 901-MR from Leica). The associated accuracy is $\pm 0.4\mu m + 0.3\mu$ m/m (error of 0.4μ m at 0 m of distance and an additional 0.3μ m for every metre of distance to the laser tracker) for the distance performance using the Interferometer (IFM) mode. In this mode, the laser measures the distance to a reflector, which can be placed in any desired position in the laboratory.

In IFM, at a 20 m distance (more than the maximum distance to be observed in the laboratory) the in-line distance and the horizontal scale bar errors are 0.0054 mm and 0.191 mm, respectively. The error increases arithmetically with distance to the laser tracker. Thus, the error of the measurements will be smaller than 0.2 mm, which is well below the 2 mm navigation requirement for the laboratory measurements as determined by Krüger and Theil [2010]. Furthermore, the angular accuracy, for the same mode is of $\pm 15\mu$ m $+6\mu$ m/m.

The IFM mode is the one used when measuring the position of reflectors, which are magnetic spheres with a piramidic reflective cut. To measure their position, the laser pointer needs to be placed on the reflective faces, which causes the reflector to be recognised by the laser tracker. The position can then be measured through the Host PC in the operations section (Section 5.1).

Another measurement method involves the Tracker-Machine (T-MAC). This extension allows to measure 6 DoF with an accuracy of $\pm 15\mu$ m $+6\mu$ m/m [1], once again sufficient for the application in study. The T-MAC will be fixed with respect to the camera and used to measure its orientation as discussed in Section 5.5.

Finally, the laser tracker also allows for the construction of DEMs through the laser-scanner. This instrument was previously used to generate the terrain models' DEMs and for a camera placement test. It can be operated through the computer found inside the simulations section.

The position of the Laser Tracker (LT) in the laboratory can be changed to more optimally align with the

[1] /https://www.hro.ipa.fraunhofer.de/content/dam/agp/en/documents/equipment/Measuring_large_dimensions/T-Probe_T-Mac.pdf, last access: 15/01/2018

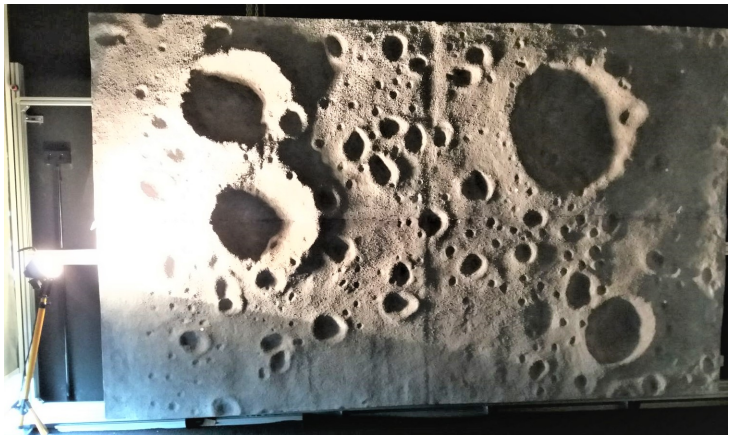


Figure 5.3: Terrain model 3

measurement ends (reflectors and/or T-MAC). Additionally, by adjusting three knobs at the base of the laser tracker, its alignment can also be changed. On the top of the instrument, there is a circular buble level, which can be used to check if the Z_{LT}-axis (Subsection 5.5.1) is along the direction of the gravity's acceleration.

5.3. Simulation Mechanism

The mechanism used for the simulation of the vehicle's dynamics is a six DoF KUKA KR 16 robot mounted on a linear rail. Thus, the whole system has seven DoF. The robot's hand holds the payload, which in this case is a plate where the camera and the T-MAC (Section 5.2) are mounted and rigidly fixed. This payload can be up to 16 kg (much larger than the required load) and move with up to 1.47 m/s of traverse velocity. The static repeatability (precision) of the robot is 0.1 mm.

The robot can be controlled by the dSPACE system, either manually or through computer scripts.

5.4. Terrain Model

Various models are availaboratoryle in TRON for simulation purposes, three of which (terrain models one to three) are used to simulate orbits and landings on the Moon and are installed in the three walls non adjacent to the operations section as indicated in Figure 5.1.

The terrain model 3, shown in Figure 5.3, will be used to acquire the data for the analysis of the software. It is about 4.2×2.2 m, has terrain dynamics (difference between lowest and highest points of the model surface) of about 26 cm and was modelled by hand with subsequent 3D scanning and post-processing. It was first milled into a rough structure; and then received a final manual finishing, which removed any visible milling lines. This finishing step provides a practically infinite resolution to the model.

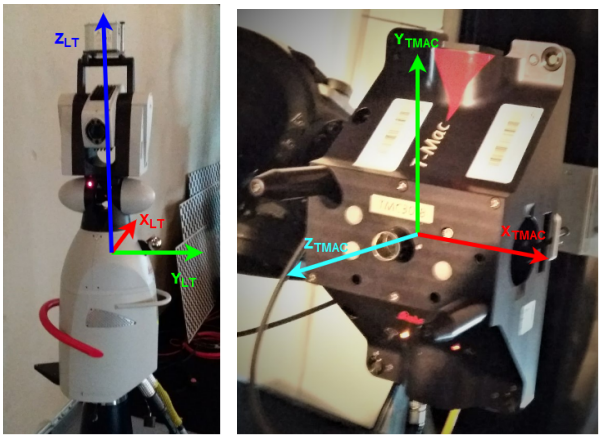
The self-similarity of the craters can be used to apply different scales to the model. For the ATON project and this research, the scale was set to 1:100 [Krüger et al., 2014]. Apart from the Moon's surface, it can also be used for asteroid simulations. It is used for TRN and HDA simulations, with optical (cameras) and 3D imaging sensors.

5.5. Referrence Frames

The measurements in the laboratory are made by the LT (Section 5.2), which can be freely positioned in the laboratory, and are given in a reference frame internal to the instrument – $\mathcal{F}_{LT}$. A DEM of the terrain model can be obtained in this frame using the laser scanner.

Since scanning a terrain model can take several hours it is preferred not to perform a scan every time the laser tracker is moved with respect to the terrain model. Instead, three reflectors are placed in a fixed position with respect to the terrain model and are used to define the model reference frame, $\mathcal{F}_M$. The terrain model's DEM is generated in this reference frame. Once the laser tracker is moved, it is sufficient to measure the three reflectors.

On the other hand, the camera pose cannot be measured directly either, since the measurements can only be made to specific instruments. To determine position and orientation, the T-MAC is used. In practice, the



(a) LT frame (b) TMAC frame

Figure 5.4: Reference frames

T-MAC is fixed with respect to the camera. The transformation between the T-MAC and the camera frames ($\mathcal{F}_{TMAC}$ and $\mathcal{F}_C$, respectively) is derived from hand-eye calibration (Section 5.6).

Upon beginning a TRON experiment, the positions of the reflectors are measured, thus defining the transformation from $\mathcal{F}_{LT}$ to $\mathcal{F}_M$. During the experiment, for each image captured by the camera, the laser tracker measures the position and orientation of the T-MAC, and thus the transformation from $\mathcal{F}_{TMAC}$ to $\mathcal{F}_{LT}$. Using the hand-eye calibration results, the camera pose is known in the $\mathcal{F}_{TMAC}$ frame and can be transformed into the $\mathcal{F}_M$ frame through the sequence described. These measurements are used both to place the camera in the 3D rendering software and to compare with the final navigation results.

The definition of the laser tracker and T-MAC reference frames are discussed in Subsections 5.5.1 and 5.5.2, respectively. The terrain model reference frame and other related frames were defined in Subsection 4.1.1.

5.5.1. Laser Tracker Frame

All the measurements performed in TRON are done so by the laser tracker and are given in a reference frame internal to it, $\mathcal{F}_{LT}$. The origin is somewhere within the instrument; the Z_{LT}-axis points up; the Y_{LT}-axis point through the front of the instrument – where there is a holder for a reflector and to where the pointer is directed in bird-bath mode (used to catch the pointer); and the X_{LT}-axis completes the right-hand rule.

This frame is used as a necessary intermediate frame between $\mathcal{F}_M$ and $\mathcal{F}_C$, and so the details of its orientation are irrelevant. Since the laser tracker can be moved and have its orientation adjusted (Section 5.2), this frame is not fixed with respect to the terrain model.

5.5.2. TMAC Frame

As with $\mathcal{F}_{LT}$, the T-MAC frame, $\mathcal{F}_{TMAC}$ is an intermediate frame required to get the camera pose in $\mathcal{F}_M$. It is thus not relevant to know its exact definition. For testing and error correction purposes, however, it is useful to have a general idea. The origin is within the instrument; the Z_{TMAC}-axis is perpendicular to the plate in which it is mounted, pointing along the antenas' direction (which must be visible by the laser tracker for a measurement to be possible); and the $X_{TMAC}Y_{TMAC}$-plane is parallel to the same plate.

5.6. Camera Calibration

When getting camera images in TRON, it is necessary to obtain the camera intrinsic matrix, the distortion parameters and the transformation between $\mathcal{F}_{TMAC}$ and $\mathcal{F}_C$. The first two are the outputs of the intrinsic calibration of the camera; whereas the last one comes from the extrinsic calibration.

Both kinds of calibration require a calibration target (see Figure 5.5) and are done through CalDe and CalLab, both software developed by DLR. CalDe is used to detect and identify the square corner points of a calibration target in an image and CalLab uses the points detected by CalDe to perform intrinsic and extrinsic calibrations. The use of these tools is discussed in Appendix A.

For good calibration results, the images should all contain the markers of the checkerboard (the three dots) and as many corner points as possible. All the images should be taken from different camera views. The

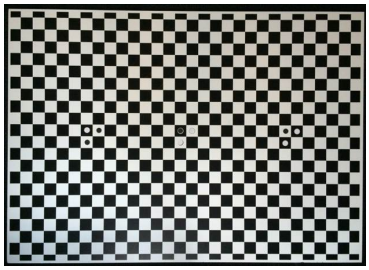


Figure 5.5: Calibration target for extrinsic calibration

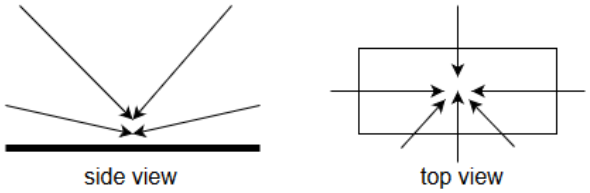


Figure 5.6: Suggested positions for camera calibration images

suggested configurations are shown in Figure 5.6. Finally, all points of the sensor should see the checkerboard at least once, meaning that every pixel should represent the checkerboard in at least one of the images.

6 | Software

In this chapter, the methods introduced in Chapters 3 and 4 are integrated into a full work-flow. The measurements obtained in the TRON laboratory (Chapter 5) were used to test and to aid the development of the software described hereafter.

This chapter begins with the software requirements, in Section 6.1. Section 6.2 In Section 6.3 the architecture of the code developed is presented, including the pre-existing software, libraries and languages used. Section 6.4 provides a brief description of the supporting software used. In Sections 6.5, 6.6 and 6.7, the acceptance, unit and system tests performed are described. Finally, the validation of the tool is discussed in Section 6.8.

6.1. Software Requirements

The requirements imposed on the software are:

- **SWR_1** – The online script shall be built using only software components and libraries for which the source code is available
- **SWR_2** – The full work-flow shall use only open-source or internal software
- **SWR_3** – The developed software shall allow for future changes and enhancement by third parties

6.2. Dataset 0

To test and aid the development of the navigation software, a preliminary dataset was obtained. It was used to test the camera placement in the 3D model and the software as it was developed. It was also used for the coarse parameter tuning (described in Chapter 7.1) and for the validation of the algorithm (Section 6.8).

This dataset includes the camera states (position and orientation) in $\mathscr{F}_M$ that comprise the trajectory, the associated online images (captured in TRON for each of the trajectory points), the positions of the three reflectors associated to the terrain model (see Subsection 4.1.1), the approximate position of the lamp and the camera calibration results (see Sections 4.2.2 and 5.6).

The trajectory used is shown in Figure 6.1, along with the image used to generate the 3D terrain model. It was a simple straight line trajectory with 61 points, starting at around 4 m and ending at around 1 m from the terrain model (see Section 5.4). The orientation of the camera remained constant, to simplify the relative positioning of the measuring instruments (see Section 5.2). The axis of $\mathscr{F}_C$ are represented for the first point of the trajectory using the *RGB* convention – red corresponds to the X_C-axis; green to the Y_C-axis; and blue to the Z_C-axis. The model frame – $\mathscr{F}_M$ – is represented with magenta for the X_M-axis, black for the Y_M-axis, and cyan for the Z_M-axis.

6.3. Architecture

The tool is composed of three major scripts, written in Python 3: script A – generate database (Subsection 6.3.1); A1 – render images and depth data (Subsection 6.3.2); and B – determine trajectory (Subsection 6.3.3). A detailed walk-through of the work-flow is given in Appendix A including the format used for the input files and the commands used to call the scripts.

6.3.1. A – generate database

Script A is the offline script used to generate the database required for navigation. This database is built using images rendered through script A1 (Subsection 6.3.2). The rendering can be done prior or during the script run, as defined by one of the inputs, represented by the conditional block *Render images* in Figure 6.2. This figure shows the architecture of the offline script.

In case the user chooses to perform the rendering, the A1 script (Subsection 6.3.2) is called using the camera parameters and trajectory files as input. Depending on the type of model selected, either Blender (Subsection 6.4.1) or SensorDTM (Subsection 6.4.2) is used to render both the images and the depth data associated to each camera pose in the trajectory file. The output files are placed in the folder chosen by the user. In case no render is to be performed, the same data should already be available in the folder specified.

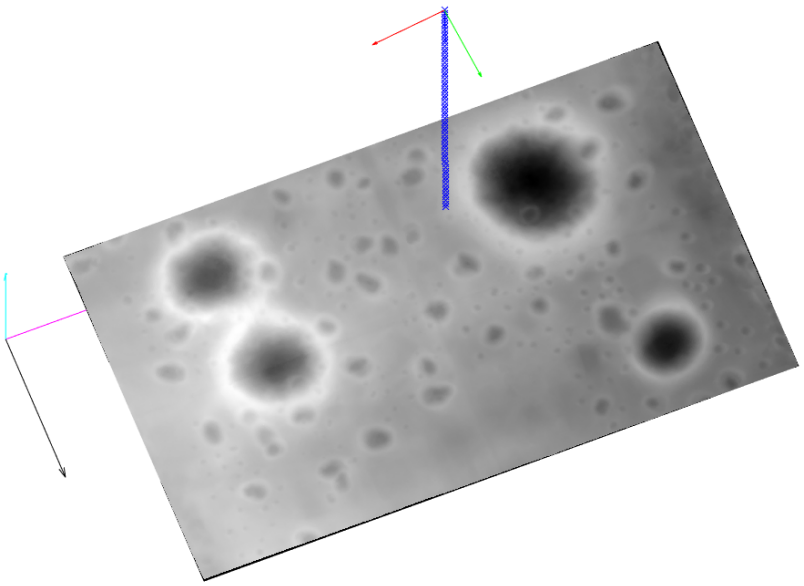


Figure 6.1: Trajectory points and camera axis used to capture the images for dataset 0

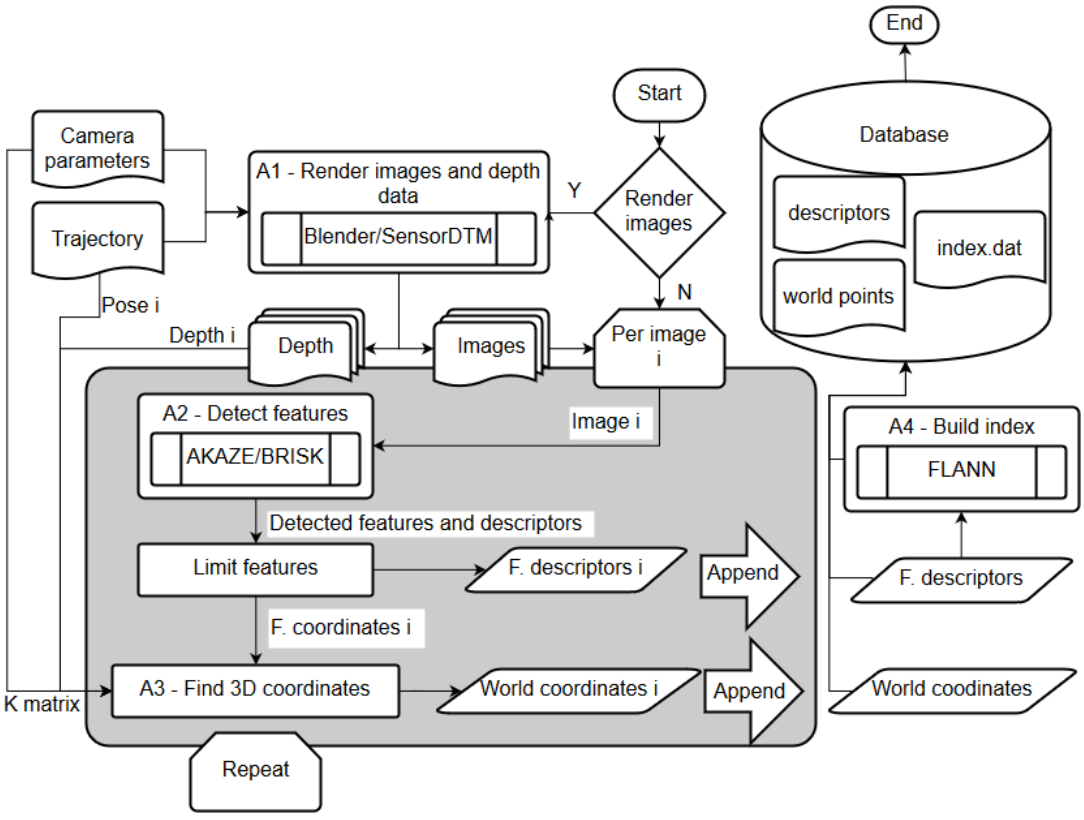


Figure 6.2: Architecture of script A – generate database

Next, for each image, the descriptors and world coordinates of the detected features are determined and appended to the appropriate lists (grey block of Figure 6.2). First, the image is read and used to detect features through the feature detector chosen by the user (AKAZE or BRISK). The result is a list of detected features and their respective descriptors which undergo a pre-selection (block *Limit features*) based on the number specified by the user (described in Section 3.5).

The descriptors of the selected features, *F. descriptors i*, are appended to the descriptors list, *F. descriptors*; whereas the associated coordinates are used as inputs to block *A3 - Find 3D coordinates*. This block uses the intrinsic parameters matrix, **K**, the camera pose and the depth data to transform the 2D image coordinates of a feature into its associated 3D world coordinates (described in Section 4.3). The output, *World coordinates i*, is appended to the end of the full list of world coordinates.

Once this process has been repeated for all images in the specified folder, the FLANN index is built using the descriptors of all the features found in the images (*F. descriptors*). The index is built by optimisation, using $w_m = 0$, $w_b = 0$, a sampling fraction of 1 and the precision set by the user (see Subsection 3.6.2). The list of feature descriptors and of world coordinates are saved as *npy* files; and the FLANN index is saved as a *dat* file. These three files comprise the database, which is the output of script A.

6.3.2. A1 – Render Images and Z-Depth

The A1 script can be called within the A script or individually and can be used to render only images or both images and the corresponding depth data. This script has two versions: one for Blender; the other for the SensorDTM.

The Blender version is a python script, which is run through the command line with Blender in the background. In other words, the 3D Blender environment is not opened or presented visually to the user, thus increasing the computation speed. The camera parameters file is used to set the necessary camera settings. For each line of the trajectory file, the camera is placed at the specified pose. Both the position and strength setting of the lamp are also changed according to the values provided. An image and, if the adequate *nodes* have been set in the model (see Apendix A), depth data are rendered into the *temp* folder. They are then renamed according to the row index of the trajectory file and moved to the appropriate folder.

The SensorDTM version is a *bat* file, which includes the relevant camera parameters (resolution and FOV). The outputs are the rendered images and, if required, the depth data, both of which are placed in the same directory as the *bat* file and named after the row index. When run through the A script, these files are automatically renamed and moved to the appropriate location.

6.3.3. B – Determine Trajectory

Script B is the online script used for navigation. The navigation solution is found using the images captured during landing and the database generated offline through the A script. This block's architecture is represented in Figure 6.3.

The index file and descriptors in the database are first used to reconstruct the FLANN index for feature matching. For every image in the specified folder, the camera pose is determined and the solution is added to the file containing the full trajectory solution (grey box of Figure 6.3).

First, as in the offline script (Subsection 6.3.1), the image is read and used to detect features, which are pre-selected. For each of the limited descriptors, a match is found via FLANN using the reconstructed index from the database. The matches found are downselected to a number given by the user (Section 3.7). The downselected matches, features coordinates found and the world coordinates from the database are inputs to determine the transformation between the current $\mathcal{F}_C$ and $\mathcal{F}_M$ frames, via the EPnP RANSAC algorithm (described in Section 4.4.1). This is repeated for every image in the folder.

6.4. Supporting Software

This section provides a brief description of some of the non-developed software used within the work-flow and to aid the algorithm's analysis. For the image rendering softwares, Blender (Subsection 6.4.1) and DLR's SensorDTM (Subsection 6.4.2), a simple description of the inputs and outputs is provided. A more in-depth explanation of Blender's use in the context of this application is given in Appendix A. Subsection 6.4.3 is an introduction to the trajectory optimiser used to generate the TRON trajectories for dataset I (see Subsection **??**).

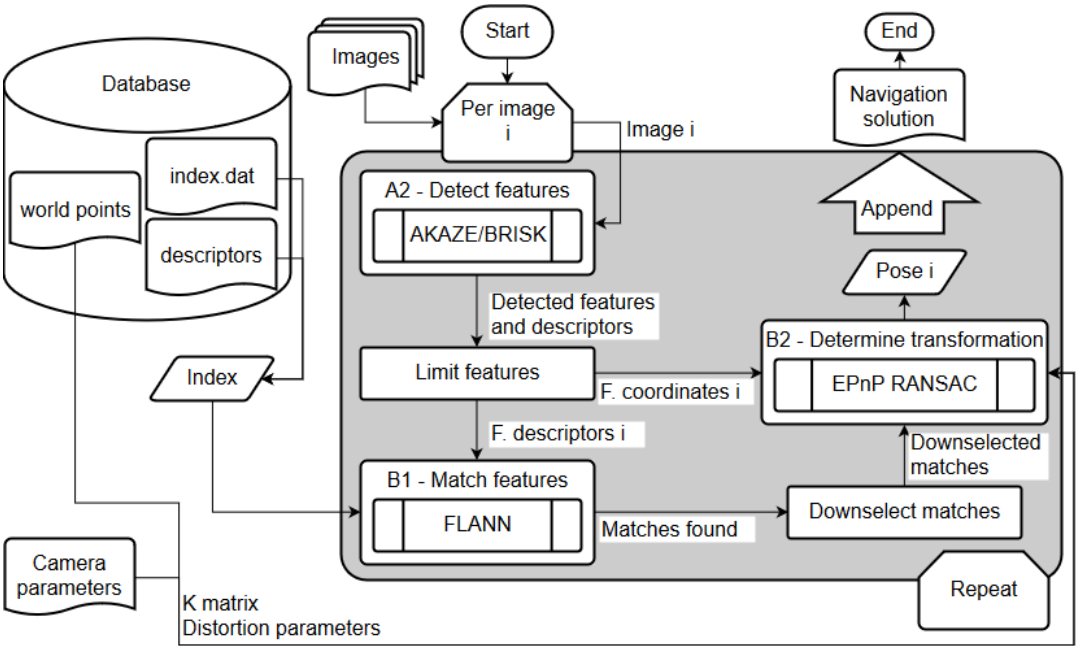


Figure 6.3: Architecture of script B – determine trajectory

6.4.1. BLENDER

Blender is a free open source 3D creation software. Among other things it can be used to create 3D models and render camera images. Scripts can be run through a python API.

It is possible to create 3D models based on DEMs. These can even be applied to spherical sections, which is of interest in the context of the Kaguya DEMs.

However, since the software is not tailored to applications that require high precision, there are some uncertainties associated with the operations. It is not clear whether the DEMs are read as pixel-is-point – each pixel of the DEM contains the value corresponding to the top right corner of that pixel – or pixel-is-area – the pixel value encodes the value at its centre. This uncertainty is not very significant for the lunar model: the Kaguya DEM has a resolution of about 7 m per pixel, corresponding to an error of about 0.35% of the LOS distance (at 2 km of altitude, lowest altitude considered) per pixel. Nevertheless, it could have an effect for the TRON model: the 1 mm resolution leads to an error of 1% of the LOS distance (at 1 m of distance to the model, lowest distance considered) per pixel. Another issue was in specifying the dynamic range when applying a DEM in a spherical section. In this context, using the displacement value associated to the DEM produced much higher deviations than expected and the value had to be reduced by around one order of magnitude.

Another functionality available is the definition of materials for the objects, including reflectivity and other texture effects. It is also possible to use images to define the objects' textures, which may be useful to render images of celestial bodies with significant surface texture.

Although distortion is not modelled, the cameras can be customised based on the desired parameters. Lighting can also be set by the user, although the parameters do not have an immediate physical counterpart and need to be set by trial and error.

Apart from rendering images, Blender can also render multiple different forms of data associated with the view of the model. Of interest is the z-depth: the distance between the camera's principal point and the object point associated with a given pixel in the image. This information can be used to determine the world coordinates from pixel points, as described in Section 4.3. To render these data, the model should include the appropriate *nodes* (more information in Appendix A). Note that, to allow for the depth data rendering through the command line, two lines of code from the Blender source file have to be changed (see Apendix A).

Provided an appropriate model, the A1 script can be run without direct interaction with Blender. As inputs, two text files need to be provided: one with the camera parameters and another with the trajectory data, including both the camera's state, the light position and strength. The outputs are the rendered images and, if desired, the z-depth data files (in the *exr* format).

6.4.2. SensorDTM

Developed by DLR, SensorDTM is an image rendering tool developed specifically for use with DEMs of celestial bodies. As such, it provides high precision in the model's placement, which is done automatically via a script (provided the DEMs and the appropriate header files). It also allows for the positioning of the Sun. The lighting can be set to full contrast – the values of the image are extremised to take full advantage of the range available – or by setting the exposure time (a more intuitive variable than the one used in Blender).

The tool uses differential rendering, meaning that it renders with as much resolution as is required (at larger distances, a lower resolution version of the model can be used), which increases the rendering speed.

As opposed to Blender, the 3D model itself cannot be viewed, making it not so intuitive to test. It also does not have any method to include textures to the models. The position of the Sun seems to only affect the light direction and not its intensity (the distance to the body has no effect). Finally, the reference ellipsoid's radius (specified in the header files) did not seem to affect the resulting model.

Apart from these aspects, the tool is not very user friendly, in that it relies on absolute paths and the outputs cannot be automatically named or placed in appropriate folders.

The depth data can also be rendered. In this case, however, it is given in the form of the z-value – the z_C-coordinate of the world point associated to a given pixel – in the *flt* format.

Using this tool through the A1 script is similar to the process with Blender, with the difference that, in the trajectory file, the quaternion entries have a different order (the scalar component is in the first position for Blender and in the last position for SensorDTM, as explained in Subsection **??**) and the last row corresponds to the exposure.

6.4.3. Trajectory Optimiser

To generate the trajectories of dataset I, used to analyse the algorithms performance (Section 7.2), the Shefex-3 Pseudospectral Algorithm for Reentry Trajectory Analysis (SPARTAN) tool was used. This tool was developed by DLR to compute the optimal trajectories for the SHarp Edge Flying Experiment-3 (SHEFEX-3) project. Since then, it has been generalised, being valid for preliminary analysis of entry, descent and landing scenarios, and having been used in various projects, for instance the Global Trajectory Optimization Competition (GTOC) [Sagliano et al., 2017].

When generating reference trajectories one of the approaches is to formulate the problem as an Optimal-Control Problem (OCP). This is an alternative to imposing a set of assumptions with the purpose of simplifying the mathematical description, thus reducing the applicability and accuracy of the solution.

This alternative approach can be formulated as such:

- given a state vector $\mathbf{x}$ (*e.g.*, a lander's state vector)
- and a control vector $\mathbf{u}$ (*e.g.*, the throttle setting)
- minimise a cost function J (*e.g.*, the consumption of fuel)
- subject to a set of differential equations (the equations of motion)
- and a set of constraints (*e.g.*, the allowed range for the control values)
- where the initial and final times and state vectors can be either fixed or free

The numerical method used to solve the OCP in SPARTAN is the flipped-Radau Pseudospectral Method (fRPM). This method is a direct method, meaning it begins by discretising the problem. In the specific case of fRPM, not only is the domain of the problem (time domain) discretised, a discrete version of the state and control vectors, as well as for the differential and integral operators, are determined.

In an earlier version of the optimiser, the discrete nodes were the roots of the flipped Legendre-Radau polynomial [Sagliano et al., 2017], resulting in p nodes. Later, the possibility to begin by dividing the time period into h segments before finding the p discrete nodes for each was added [Sagliano, 2018]. In either case, the resulting non-uniform grid of nodes maps the time into a pseudo-spectral time. The relation between the two is then used to determine the conversion for the state and control vectors; and the differential and integral operators in their discrete version. With the fRPM, the initial state is a required input.

Unlike methods employing equi-spaced nodes, an increase of the number of nodes used in fRPM, as with other pseudo-spectral methods, increases the accuracy of the result. This method also yields smoother results that require a smaller number of nodes to generate an acceptable solution. The new method introduced in SPARTAN includes various novelties that significantly reduce the computation times and improve the condition number of the problem [Sagliano et al., 2017].

However, it is usually hard to use in real-time (high computational times); could yield local optima; and it may require a good initial guess. Due to the density of the matrices involved, fRPM may be slower than the standard methods for higher number of nodes. These were the reasons that motivated the combination of the

fRPM method with convex optimisation and the hp scheme (segmenting the time domain before discretisation as described above) [Sagliano, 2018].

When using the hp scheme, the problem is reformulated into h similar sets of equations, each one related to a set of state variables unique to that segment. As a result, the matrices involved are now quasi-block-diagonal, thus greatly reducing the computation time required. Apart from the segmentation of the time domain and the state vector, a new set of constraints – the linking constraints – are added, which guarantee the continuity between the segments: the initial time and state of segment j is equal to the final ones of segment $j-1$.

On the other hand, a method that can be used in real-time applications is convex optimisation. A convex optimisation problem is defined as:

- minimize $J = f_0(x)$
- subject to $f_i(x) \leq a_i, i = 1, ..., m$
- where $f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y), i = 0, ..., m$
- with $\alpha \geq 0, \beta \geq 0$

This method of trajectory generation is of increasing interest given that many problems can be convexified, efficient solving methods exist, no initial guess is required, and a global maximum is guaranteed when the problem is feasible. SPARTAN uses the Second-Order Core Programming (SOCP) method to solve the convex optimisation problem.

A comparison of this method (using both the p and hp schemes) with the standard method found it more accurate (where accuracy was measured as the difference between disturbed and nominal trajectories). Of the two schemes, the p scheme was the most accurate. However, the hp scheme resulted in computation times adequate for real-time use, namely 13 times less than the p scheme when using 100 nodes [Sagliano et al., 2017]. Although the standard method is still faster, the gain in accuracy with the new method does not come at a significant increase in time cost.

Sagliano [2017] performed an analysis of the effect of the number of nodes using the Mars Science Laboratory (MSL) descent phase. The conclusion was a much higher accuracy with little time cost when using the new method, for which the range of 40-50 nodes was associated with the most gain. The characterisation of computation times was made by repeating the runs 10 times and using the average. The fRPM method was also successfully tested using the Space Shuttle reentry, the Curiosity descent phase and an asteroid landing study by JAXA-DLR [Sagliano et al., 2017].

6.5. Acceptance Tests

As indicated in Section 6.3, the proposed tool will use different available software, namely Blender and the DLR-internal tool: SensorDTM (for image rendering); and the OpenCV implementations of AKAZE and BRISK (for feature detection and extraction), FLANN (for feature matching) and EPnP (for camera pose calculation). To justify using these tools, they were first tested to determine their applicability. A summary of the acceptance tests performed is shown in Table 6.1.

Each of the following subsections details the procedure and results of these tests.

6.5.1. Test 1 – Blender General Use

This test was designed for familiarisation with the rendering procedure and the effects of moving the camera in the rendering result. Table 6.2 provides an overview of the camera positions used for this test and the expected results. The 3D model used was downloaded [1] and the result image provided by the authors is shown in Figure 6.4a.

The original image seems to have a different tonality and be darker around the objects than the result image 1 (Figure 6.4b). Since the original image was rendered in 2015, it is possible that new updates of the software have improved the render quality. The remaining tests show the results expected (Figure 6.5).

6.5.2. Test 2 – Image Rendering

This test was designed to check for consistency between the camera pose and the rendered images. To do so, a simple model comprised of three cubes (one with four Blender units of size; another with two; and another with one) shown in Figure 6.6. The cubes were created and placed using the python API to guarantee their correct placement. The cameras were placed in nine different poses, which are summarised in Table 6.3, also

[1] /http://www.eofw.org/bench/, last access: 05/01/2018

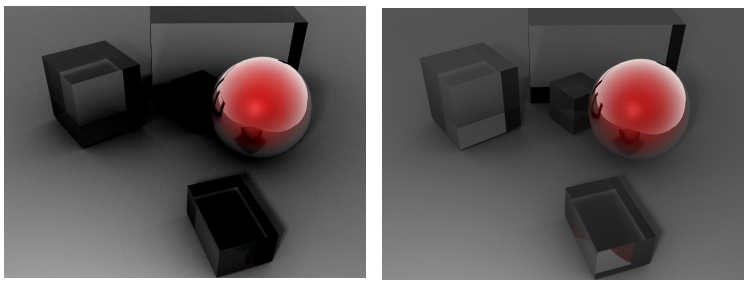
[2] /http://www.sgidepot.co.uk/perfcomp_RENDER1_blender.html, last access: 05/01/2018

Table 6.1: Summary of acceptance tests performed

Test number	Software	Functionality tested	Objective
1	Blender	General use	Familiarity
2	Blender	Image rendering	Building simple models Placing camera through Python API Determining (qualitatively) if rendered images are consistent with camera placement
3	Blender	Z-depth	Setting render nodes Determining if z-depth values are correct
4	Blender	Object placement	Determining if model and camera are placed correctly with respect to one another
5	FLANN	Data matching	Check if correct NN solution is given
6	AKAZE/BRISK	Feature detection	Check repeatibility of feature detectors
7	AKAZE/BRISK	Feature matching	Check repeatibility of feature descriptors
7	AKAZE/BRISK Blender	Feature matching	Check repeatibility of feature descriptors Check if rendered images are representative of real images
8	EPnP	Camera pose calculation	Check if correct camera pose solution is given
9	SensorDTM	Z-values	Determining if z-values are correct

Table 6.2: Overview of the test images rendered for the acceptance test 1 – Blender general use

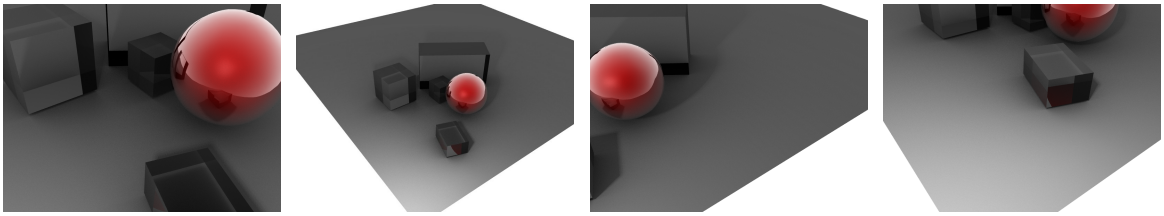
Image number	Changes made in model or environment	Expected result
1	None (original set-up)	Equal to image presented in the Figure 6.4a (original)
2	Camera moved closer to model along principal axis	Effect similar to zoom in on model (apart from perspective effects)
3	Camera moved farther from model along principal axis	Effect similar to zoom out from model (apart from perspective effects)
4	Camera moved along its X-axis to the right	Effect similar to translating original image to left (apart from perspective effects)
5	Camera moved along its Y-axis down	Effect similar to translating original image up (apart from perspective effects)



(a) Original rendered image[2]

(b) Result of test image 1; expected: same as original

Figure 6.4: Comparison between image provided by authors and image rendered using the downloaded model



(a) Result of test image 2; expected: zoom in

(b) Result of test image 3; expected: zoom out

(c) Result of test image 4; expected: translation left

(d) Result of test image 5; expected: translation up

Figure 6.5: Results for test images 2 to 5

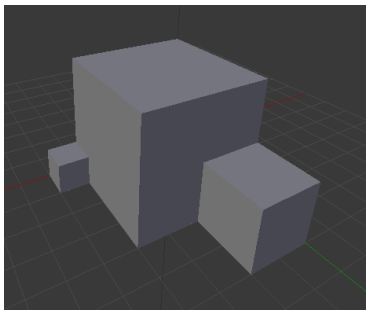


Figure 6.6: Cube model used for test 2 – image rendering

through the python API. To note that the Blender camera reference system has the Y_C- and the Z_C-axis in the opposite direction as what is used in the pinhole camera model (Subsection **??**).

The results are shown in comparison with the expected results in Figure 6.7. Before performing the test, the images representing the expected results were created with simple image editing software. They were created considering only the orientation of the camera with respect to the cubes and orthographic projection (thus, they do not attempt to represent the perspective effects). The objective was to represent the relative position and orientation of the cubes in the image. Figure 6.7o was obtained by rotating Figure 6.7m by 45° in the direction opposite to the rotation made to the camera. Figure 6.7q is a translated version of Figure 6.7g.

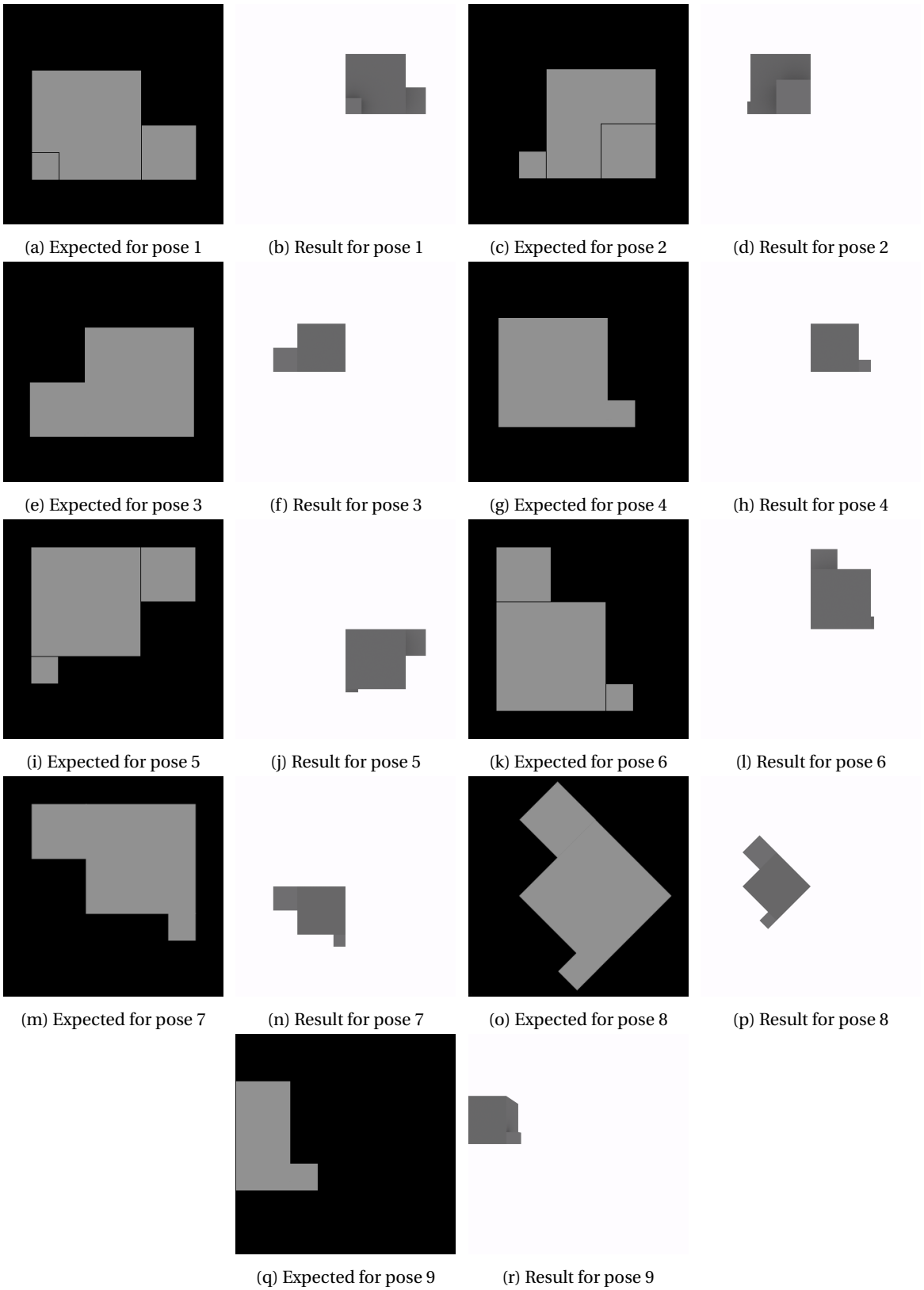
The rendered images are consistent with the prior expectations. The differences observed in the relative sizes of the cubes and the side faces visible in Figure 6.7r are consequences of the perspective effects, ignored when creating the expected images.

6.5.3. TEST 3 – Z-DEPTH

Apart from the camera images used for the database, the z-depth associated with each one of them is also rendered. This information is saved by Blender in the form of an OpenEXR file, in the form of 16-bit or 32-bit floats. The following test was designed to determine whether the information saved truly represents the distance from the camera to the world point that is projected into the coresponding pixel image.

Table 6.3: Camera poses used for acceptance test 2 – image rendering

Pose Number	x	y	z	q_0	q_1	q_2	q_3
1	20	0	0	0.500000	0.500000	0.500000	0.500000
2	0	20	0	0.000000	0.000000	-0.707107	-0.707107
3	-20	0	0	0.500000	0.500000	-0.500000	-0.500000
4	0	-20	0	0.707107	0.707107	0.000000	0.000000
5	0	0	20	0.707107	0.000000	0.000000	0.707107
6	0	0	20	1.000000	0.000000	0.000000	0.000000
7	0	0	-20	0.000000	0.707107	-0.707107	0.000000
8	0	0	-20	0.000000	0.382683	-0.923880	0.000000
9	10	-20	0	0.707107	0.707107	0.000000	0.000000



(a) Expected for pose 1 (b) Result for pose 1 (c) Expected for pose 2 (d) Result for pose 2

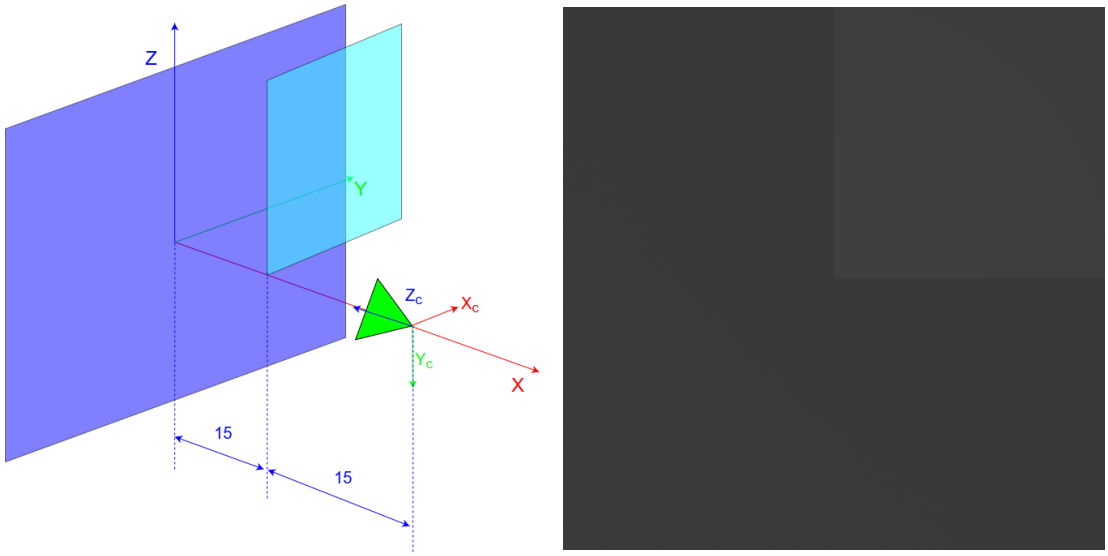
(e) Expected for pose 3 (f) Result for pose 3 (g) Expected for pose 4 (h) Result for pose 4

(i) Expected for pose 5 (j) Result for pose 5 (k) Expected for pose 6 (l) Result for pose 6

(m) Expected for pose 7 (n) Result for pose 7 (o) Expected for pose 8 (p) Result for pose 8

(q) Expected for pose 9 (r) Result for pose 9

Figure 6.7: Comparison between expected and obtained results for test 2 – image rendering



(a) Squematic of z-depth test set-up

(b) Image rendered for test 3 – z-depth

Figure 6.8: Z-depth test set-up

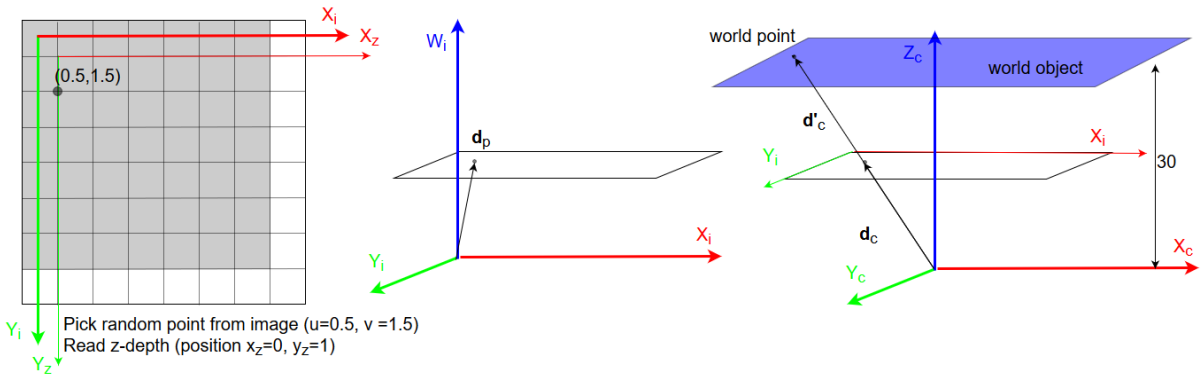


Figure 6.9: Geometric representation of the method used for test 3 – z-depth

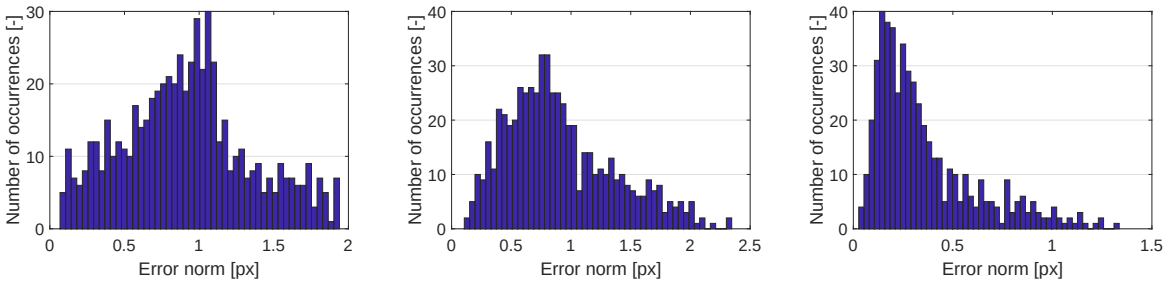
The camera was placed at $(30,0,0)$ oriented towards the origin (quaternion equal to $\begin{pmatrix}0.5 & 0.5 & 0.5 & 0.5\end{pmatrix}^T$). The model was composed of two planes: one at $x = 0$; the other at $x = 15$, for $y > 0$ and $z > 0$. Figure 6.8a shows a squematic of this configuration. Note that in the image the camera reference frame represented is the one used in the pinhole camera model, not the one used in Blender. Figure 6.8b is the rendered image.

As can be recognised through these images, in the camera's reference system, the points visible either have $z_c = 15$, when in the top right quadrant of the rendered image – $u > 511$ and $v < 512$; or $z_c = 30$ otherwise. The test performed is described below for each of the $N = 1000$ points considered:

1. Pick a random point of the image (note that to get the centre of the pixel, 0.5 was added to the integer values of each coordinate)
2. Define the vector pointing to the chosen pixel in the image reference frame in homogeneous coordinates, $\mathbf{d}_p$
3. Transform the vector into the camera reference frame, $\mathbf{d}_c$, using the inverse of the intrinsic camera matrix
4. If $u > 511$ and $v < 512$, scale the vector so that the third coordinate equals 15
5. Otherwise: scale it to 30
6. Calculate norm of the vector, d
7. Compare the result with the z-depth stored in the pixel considered

This method is represented in Figure 6.9.

The error, calculated as the difference between d and the z-depth associated with the generated pixel, was in the order of 10^{-6}. Thus, it can be concluded, that the data saved as the z-depth is the expected distance



(a) Original image (b) Undistorted image (c) Rendered image

Figure 6.10: Error norm histograms for first image of the test set

between the camera and the world point associated with the centre of each pixel of the rendered image.

6.5.4. Test 4 – Object Placement

This test was designed to determine whether the relative positioning of a model and a camera in Blender was correct. To perform this test, a work-flow developed during the internship period was used. This requires a data file containing the checkerboard square corner points in the associated reflectors frame, $\mathbf{c}^I$, which was generated during the internship.

The following procedure is used:

- In TRON:
  1. Place checkerboard and measure the reflectors' position
  2. Orient the robot arm, with the camera, making sure the circles of the checkerboard are visible in the image
  3. Capture image and measure associated T-MAC pose
- Determine camera pose(s) in the checkerboard reference system and render corresponding images in Blender
- Determine error between points in the image and the projected corner points (for the original, undistorted and rendered images):
  1. Project checkerboard corner points onto image $\mathbf{c}^i$
  2. Use CalDe to find the real corner points in the image
  3. Plot histogram of the error (norm of the vector difference between the real and projected corner points in pixels)

Figure 6.10 shows the comparison between the histograms of a typical image for the original, undistorted and rendered versions.

Since the errors for the rendered image are smaller than for the remaining ones, the placement of the objects in Blender can be considered correct. The errors that exist can be due to errors in the initial determination of the checkerboard points in the reflectors frame. The additional errors may be caused by camera calibration errors.

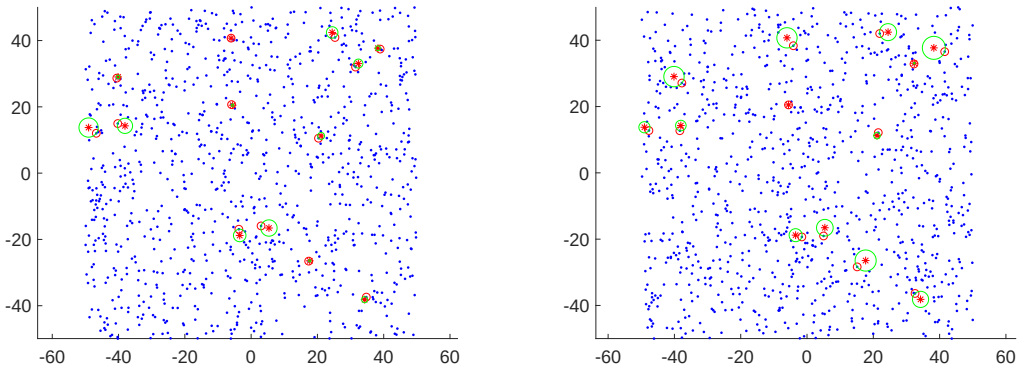
6.5.5. Test 5 – Data Matching

To accept FLANN as an appropriate tool to find matches between descriptors, two tests were performed: one using 2D points; the other using 128D points (vectors with 128 entries). For the 2D test, the following procedure was used in three different runs.

- Generate dataset of 1000 random 2D points with coordinates between -50 and 50
- Generate 10 random 2D points to use as queries with coordinates within the same limits
- Generate optimised index for fastest search ($w_b = 0$, $w_m = 0$), target precision of 90% and a sampling fraction of 1
- Find NN of each query
- Plot datapoints, query points and respective matched points and distance circles

A match is correct if the minimum distance circle does not contain any data point inside. Figure 6.11 shows the resulting plots for the first two runs.

From the three runs, only one point in the first run has been mismatched, namely with the second closest point rather than the closest. The match precision is above the precision chosen when optimising the index.



(a) Run 1

(b) Run 2

Figure 6.11: 2D FLANN matching results plots; blue dots – data points; red stars – query points; red circles – matched points; green circles – minimum distance circles

For the 128D test:

- Generate dataset of 1000 random 128D points with coordinates between -50 and 50
- Choose 10 points of the dataset at random to use as queries
- Generate optimised index for fastest search ($w_b = 0$, $w_m = 0$), target precision of 90% and a sampling fraction of 1
- Find NN of each query
- Compare id of the queries to the id of the match

A match is correct if both ids are the same, which was observed in the test made.

6.5.6. Test 6 – Feature Detection

To test the repeatibility of the feature detection for the important transformations (namely, translation, rotation, zoom and illumination changes), the following test was performed. The image 8 from the CE-3 mission was cropped to exclude the spacecraft structures and the area above the horizon. This cropped image was then used to derive a total of 36 test images. For each of the images, the following procedure was taken:

- Detect features in original image
- Detect features in transformed image
- Transform coordinates of the transformed features into the original frame
- For every pair of original and transformed features, calculate distance
  - If distance is below a given threshold, calculate hamming distance between descriptors
    - If hamming distance is below previous one (indicating that a better match has been found), save hamming distance and match (overriding any previous match with repeating features
- Calculate percentage of matches as ratio between number of matches found and the minimum between original and transformed features

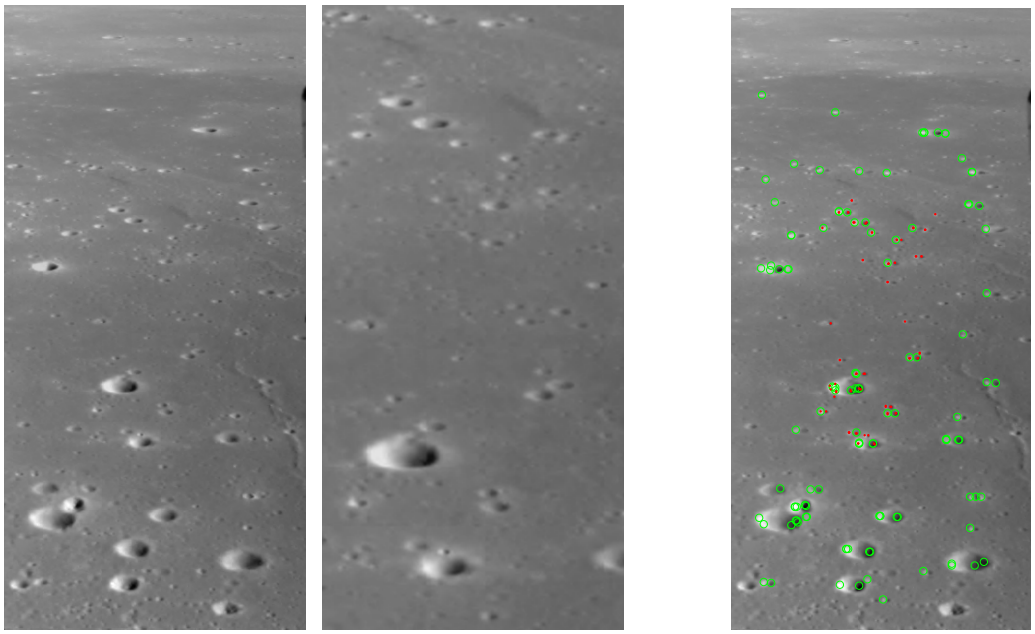
The method used to quantify the percentage of matches is only truly applicable for the changes in illumination, since the area of the image that is evaluated is constant. For the remaining ones, this quantification is flawed, given the impossibility of overlap in certain areas of the images. Table 6.4 shows the transformations considered and the results obtained for AKAZE and BRISK. The distance threshold considered was 2 px, to account for the errors in the transformation.

Changes in illumination are handled well for both detectors, with repeatibility above 85%, apart from the -100 contrast by BRISK. For the translation and rotation, despite the existence of non overlapping area between the two images compared, the repeatibility is never lower than 60%. The detection only breaks this limit for the stronger variations in zoom, for which the area difference is most significant. Figure 6.12 shows the example for the worse result (200 % zoom with AKAZE).

This example clearly shows the low value observed is a consequence of the area difference. The test allows the acceptance of both feature detectors as providing sufficient repeatibility for the required transformations.

Table 6.4: Image transformations and results of test 6 – feature detection; brightness and contrast refer to *IrfanView* settings

Transformation	Value	Results (%)		Transformation	Value	Results (%)	
		AKAZE	BRISK			AKAZE	BRISK
Brightness	25	100.0	96.77	Contrast	25	98.21	98.62
	50	100.0	95.95		50	99.11	98.17
	100	98.15	95.45		75	91.96	97.25
	150	100.0	88.89		100	81.25	86.70
	-25	100.0	96.77		-25	95.40	95.00
	-50	100.0	95.95		-50	100.0	95.24
	-100.0	98.15	95.45		-75	97.50	86.49
	-150	100.0	88.89		-100	88.24	64.71
Translation	(0,100) px	94.64	78.90	Zoom	50 %	48.94	53.07
	(100,0) px	72.12	63.10		75 %	58.62	56.88
	(100,100) px	67.89	60.00		80 %	63.16	59.17
Rotation	10°	85.58	75.60		90 %	79.63	68.91
	20°	83.70	75.69		110 %	80.56	81.07
	30°	71.43	67.31		120 %	77.32	76.98
	45°	65.82	67.67		130 %	76.67	83.78
	90°	97.32	92.63		150 %	55.56	81.19
	180°	92.86	75.47		200 %	47.67	91.84



(a) Original image (b) Transformed image (c) Result; green circles – features found in original; red dots – features found in transformed

Figure 6.12: Test result for image zoom of 200% using AKAZE

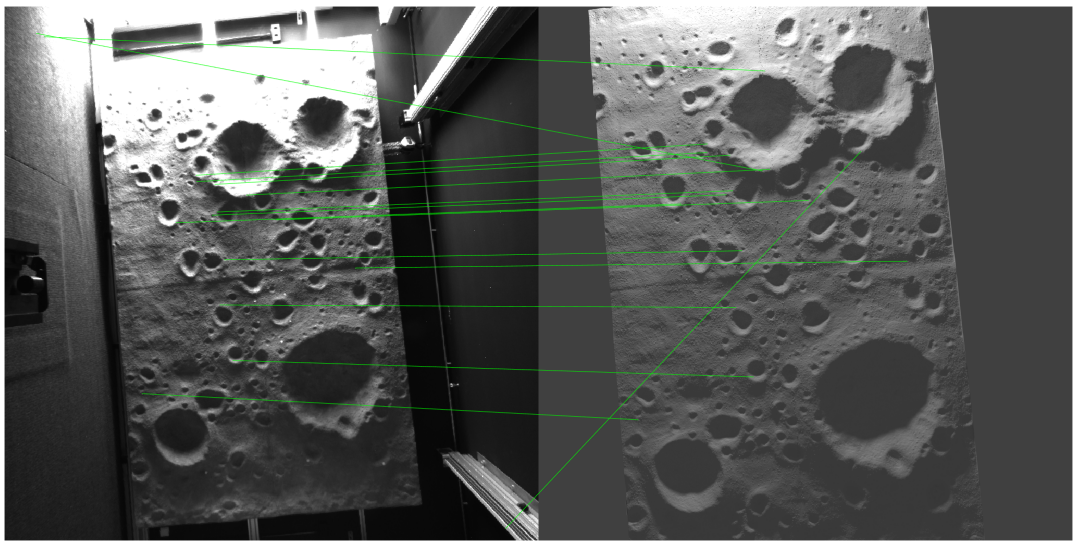


Figure 6.13: Selected matches found between first input image and second data image

6.5.7. Test 7 – Feature Matching

To perform this test on feature descriptor repeatibility and the applicability of Blender to render images representative of TRON's terrain model 3, the verified block A (see Subsection 6.7.1) was used to generate the database. First, the camera was calibrated and five images were taken in TRON, along with the respective T-MAC measurements and the position of the model's reflectors. These measurements were used to position the terrain model and the camera in the Blender environment and to define the camera's parameters. The real images were also undistorted.

The undistorted images and the database were given as input to a test script including blocks A2 and B1. The matches were reduced to the best unique matches, meaning each database feature can only be paired with one feature of the input image and vice-versa. From the unique matches, the best N (an algorithm parameter) are chosen based on the similarity between the descriptors (calculated during the feature matching).

Figure 6.13 shows the worse matching observed, namely for the first image using BRISK. Most of the matching errors are caused by the surrounding environment seen in the real images, which is not representative of what would be seen during a lunar landing and which can be corrected for future testing. Additionally, given the intention to use RANSAC, it is expected that the worse matches will end up being discarded.

6.5.8. Test 8 – Camera Pose Calculation

To test the EPnP algorithm for pose estimation, the following test was performed, using the camera poses from Table 6.3:

- Generate 50 random world points with coordinates ranging from -5 to 5
- For every camera pose:
  - Project world points to image coordinates
  - Calculate camera pose with EPnP RANSAC using the 3D and corresponding image coordinates as matches
  - Compare with original camera poses

No distortion was considered and the camera matrix used was

$$\mathbf{K} = \begin{bmatrix} 775 & 0 & 512 \\ 0 & 775 & 512 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

Up to a precision of six decimal digits, the quaternion calculated was exact and the position had a maximum error of 1×10^{-6}. Thus, it can be concluded, that the EPnP algorithm associated with RANSAC serves the intended purpose.

Table 6.5: Results of test 9 – z-values

Latitute [deg]	Longitude [deg]	$z_{C'}$ [m]	z-value
43.5	340.5	-2588.4	-2588.8
43.49	340.5	-2591.6	-2591.8
43.51	340.5	-2586.8	-2587.4
43.5	340.49	-2589.6	-2589.8
43.5	340.51	-2588.0	-2588.1

Table 6.6: Summary of unit tests performed

Block ID	Block name	Test objective
A1	Render images and z-depth	Check automatic camera placement and data rendering
A3	Calculate world coordinates	Check calculation results
-	Limit features	Check features chosen
-	Downselect matches	Check matches chosen

6.5.9. Test 9 – Z-values

To test the values stored in the *flt* files by the SensorDTM renderer, the camera was placed at the centre of one of the tiles used to build the 3D model (at 43.5° latitude and 340.5° longitude) and at a radial distance equal to the lunar radius (1737400 m), corresponding to

$$\mathbf{T} = \begin{pmatrix} 1187978.5 & -420685.2 & 1195947.2 \end{pmatrix}^T \tag{6.2}$$

The camera was oriented such that the Z_C-axis was pointing towards the Moon's centre and the X_C-axis was in the direction of X_M-axis and perpendicular to Z_C. The resulting quaternion used in SensorDTM (already transformed into the $\mathscr{F}_{C'}$ frame) was

$$\mathbf{q'} = \begin{pmatrix} 0.1565 & 0.3624 & 0.06188 & 0.9167 \end{pmatrix}^T \tag{6.3}$$

The camera parameters used were the same used to render the CE-3 images, namely resolution of 1024, $f = 8.3$ mm and pixel size of 6.7 μm.

Five points were picked by choosing their latitude and longitude, so that they were visible in the rendered images. The coordinates were transformed into Cartesian coordinates, using the altitude data from the same DEM used for the SensorDTM 3D model. They were then transformed into the $\mathscr{F}_C$ frame (by pre-multiplying with the extrinsic parameters matrix **E**) and projected onto the image. The resulting image coordinates were first used to retrieve the z-value associated, which was then compared to the respective z_C-coordinate of the point. The results are expressed in Table 6.5

As can be seen, the difference in values is never above 1 m. Considering the discrete nature of both the z-value data and the DEM used to calculate the world coordinates from the latitude and longitude, this error is within the acceptable range.

6.6. Unit Tests

In this section, the individual blocks developed will be verified. The tests performed are summarised in Table 6.6. Each of these tests are described in the following subsections.

6.6.1. Unit Test A1 – Render Images and Z-depth

This block can be used to render either images or both images and their corresponding z-depth. The switch between the two options was tested successfuly.

To test whether the automation of the image and z-depth rendering is being made correctly, namely the placement of the camera and the definition of its parameters based on text file data, the same cube model, camera poses (defined in Table 6.3) and parameters were used. A positive test would yield the same outputs as obtained in Subsection 6.5.2.

The rendered images were the same as in the previous set, thus confirming the correct placement of the camera and the setting of its parameters. To test the z-depth, the values for the files obtained in Subsection

Table 6.7: Expected results for A3 – calculate world coordinates verification test

Condition	$0 \leq y \leq 1 \wedge 0 \leq z \leq 1$	$4 < y \leq 6 \wedge 0 \leq z \leq 2$	otherwise
Expected x value	5	2	4

Table 6.8: Results for A3 – calculate world coordinates verification test

u (px)	v (px)	x	y	z
586.5	501.5	4.000000	1.729464	0.243750
530.5	384.5	4.000000	0.429464	2.959822
556.5	473.5	4.999996	0.993303	0.859375
564.5	464.5	3.999996	1.218751	1.102678
670.5	357.5	3.999998	3.679464	3.586607
593.5	465.5	4.000000	1.891965	1.079464
607.5	345.5	3.999998	2.216965	3.865179
553.5	467.5	5.000000	0.926340	0.993303
700.5	459.5	1.999996	4.712501	1.312500
621.5	470.5	4.000000	2.541964	0.963393

6.5.2 and the ones from the current test with the z-depth rendering switch on were read. The values were subtracted, squared and added. In case the data is the same between two files of a pair, this sum should be close to zero. The maximum observed error was in the order of 10^{-5}, which is in accordance to the expected.

6.6.2. Unit Test A3 – Calculate World Coordinates

The A3 block needs to be separately verified for the z-depth and the z-value data.

For the first case, the data from the result of acceptance test 2 – image rendering (see Subsection 6.5.2) was used, namely the z-depth obtained for the first camera pose. Ignoring any point belonging to the background of the image (not part of the model), ten random points were picked. These points were given as inputs to the block and the results inspected. The expected results are expressed in Table 6.7.

The results can be seen in Table 6.8, which have errors in the order of 10^{-6} with respect to the expected values.

The z-value case was verified using data from the acceptance test 9 – SensorDTM (Subsection 6.5.9). The image coordinates of the points were given as inputs to A3 to compute their world coordinates. The results were compared to the real world coordinates. The maximum error observed was of 1 m in the y_W-direction for the fifth point (latitude of 43.5 and longitude of 340.51). This is less than 0.04% of the LOS distance (around 2588 m) and is, thus, an acceptable error for the application.

6.6.3. Unit Test – Limit Features

To test the feature pre-selection algorithm (described in Section 3.5), block A2 was used to detect features in an image which were then limited to 500 features through the block *Limit Features*. The coordinates of the features both before and after the pre-selection were written into files. For the unlimited features, the response value was also included. This was repeated for the same image with n_{div} set to 1 (no divisions) and 3 (image divided in 9 equal squares: 3 rows and 3 columns).

When $n_{div} = 1$, the features are selected only based on the response value (since dividing an image in one row and one column leads to a single area containing the entire image). Thus, the features selected with this setting will be the features with the highest response values from all the ones found in the image.

On the other hand, when $n_{div} \neq 1$, the selection is distributed equally through the n_{div}^2 areas in which the image is divided. In other words, even if all the features in a given image area have a smaller response than the ones from another area, some of them ($n_{f/div}$, see Section 3.5) will still be included.

As expected, the unlimited features for both values of n_{div} were the same, and the number of limited features was 500 in both cases. To check if the features being chosen were the ones with higher response values, the following procedure was used for the case with $n_{div} = 1$:

- For each limited feature
  - Find the maximum response and corresponding list index in the unlimited features

- Check if corresponding unlimited feature coordinates are the same as the current limited feature ones
- Remove the maximum response from the list

No differences between the coordinates of the limited features and the unlimited features with maximum response values were found.

To check the effect of the image divisions in feature selection, the limited features for both cases were ploted with a grid representing the division expected with $n_{div} = 3$. It was observed that, in this case, for some divisions with less than 56 points (corresponding to $n_{f/div}$), all points were selected, whereas none had been selected for $n_{div} = 1$. This shows that, despite having a lower response value than the features in other image areas (since they were not selected with $n_{div} = 1$), they were selected as a consequence of the image division imposed.

The algorithm is, thus, considered verified.

6.6.4. Unit Test – Downselect Matches

For this test, a database was generated with script A, which had already been verified (Subsection 6.7.1). The limited features resulting from the unit test described in Subsection 6.6.3 with $n_{div} = 1$ were used to find the matches to this database. The id of the matched database point, the match distance and the feature coordinates were written to a file. Then, the matches were down-selected with the block under test and the same variables were outputed to a new file. This was done with the match limit set to 100.

The number of down-selected matches were, as expected, 100. To check whether the chosen matches corresponded to the ones with the least matching distance, a procedure similar to the one of Subsection 6.6.3 was used, this time with the minimum distance (instead of the maximum response). The compared variables were the matching distance, the database match point id and the feature coordinates.

Some differences were detected in these last two variables. However, this was due to the fact that multiple matches had the same matching distance, and the down-selected points were ordered based on the database match point id, whereas the initial matches were not. As a result, the matches with the same distance did have a correspondence between the two sets, but were shuffled with respect to one another.

In conclusion, the down-selection algorithm works as expected.

6.7. System Tests

In this section, the tests for the offline and online scripts are described (Subsections 6.7.1 and 6.7.2, respectively).

6.7.1. Unit Test A – Generate Database

The purpose of this test was to check whether the integration of the individual blocks was done correctly. To do so, the same input was used to generate two different databases: the first, by manually transfering the data from one block to the next; the second, through the integrated A block. The comparison was made by checking the sum of the square of the difference between both the world points and the descriptors of the features found. In both cases, this value was zero, meaning the outputs are the same and the correct integration was achieved.

Since each of the individual blocks were previously verified, this test is sufficient to verify the integrated block.

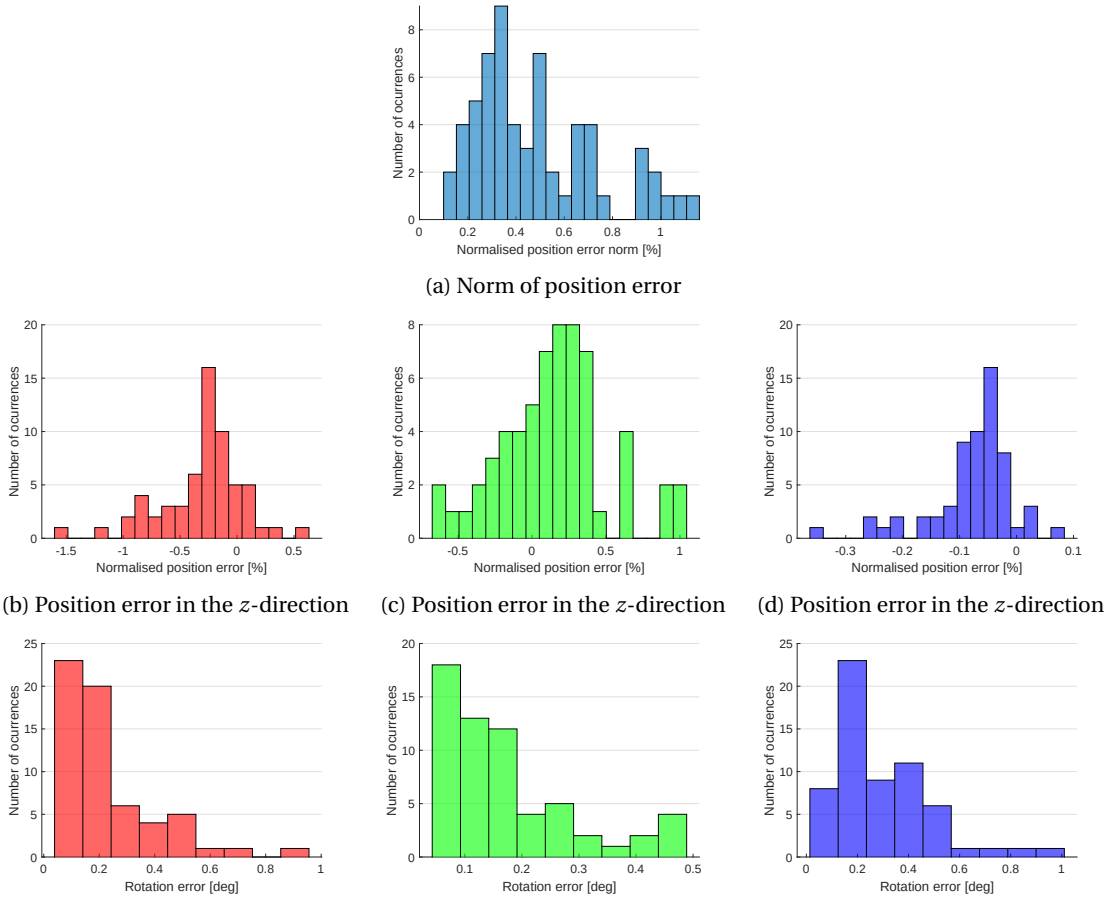
6.7.2. System Test B – Calculate Camera Poses

As in Subsection 6.7.1, to verify the B block, it is sufficient to compare the output of two runs (one with manual data transference, the other through the integrated block) using the same inputs. The database was generated using the verified A block and the image used was the undistorted version of a real image captured in TRON. The outputs were the same.

Given that the three individual blocks (A2 – process image; B1 – match features; and B2 – determine transformation) had already been accepted, the B block can be considered verified.

6.8. Validation

To validate the navigation algorithm, dataset 0 (described in Section 6.2) was used. The process detailed in Appendix A was followed to generate the appropriate Blender model, the trajectory and camera parameters files, the database (rendered) images and depth data. These data were used to generate the database. The original



(a) Norm of position error

(b) Position error in the z-direction

(c) Position error in the z-direction

(d) Position error in the z-direction

(e) Orientation error of the x-axis

(f) Orientation error of the y-axis

(g) Orientation error of the z-axis

Figure 6.14: Error histograms for navigation solutions using the tuned AKAZE parameters on the original images from dataset 0

images captured in the lab (before undistortion) were used to obtain the navigation solution. The algorithm parameters used were obtained from the tuning process described in Chapter 7.1 and are summarised in Table 7.5.

The results were used to analyse the accuracy of the navigation solution. The accuracy results are shown in Figure 6.14 in the form of histograms: Figure 6.14a for the position error norm; Figures 6.14b through 6.14d for the position error in each world coordinate (difference between real vector and calculated one); and Figures 6.14e through 6.14g for the orientation errors of each axis. The position errors are given as a percentage of the LOS distance. The orientation errors are defined as the angle between the real and calculated camera axis in question.

For an estimate of the time required per pose, the online script was run 10 times. The average time required was 0.7 s. Figure 6.15 shows the average time distribution.

The navigation error had an average below 1% and a maximum error below 3%, thus fulfilling the requirements imposed (see Section 2.5). The angle errors were lower for the y-axis and around the same magnitude for both the x- and z-axis. As for the position errors, it was lower in the z-direction and of the same magnitude for the remaining ones.

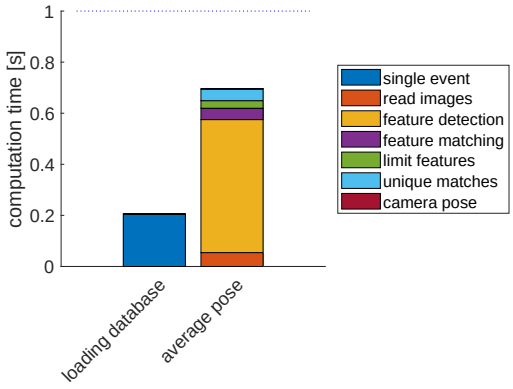


Figure 6.15: Average time distribution for 10 runs of online script using the tuned AKAZE parameters on the original images from dataset 0

7 | Results

The final results of this research, applying the work-flow described in Chapter 6 with the data obtained in the TRON laboratory (Chapter 5) and available for the CE-3 mission (Section 2.4), will be given in this chapter.

First, in Section 7.1, the algorithm parameters are tuned based on a grid search. In Section 7.2, the datasets generated in TRON and the results obtained with these datasets are discussed. The image representability of the rendered images for the CE-3 mission are discussed in Section 7.3.

7.1. Algorithm Tuning

While choosing and learning about the individual components used to build the navigation software, some of their aspects were suspected to impact the final accuracy and computation time of the navigation algorithm.

For example, it was observed (Section 3.7) that the match performed by FLANN may return duplicated and incorrect matches. Since these matches are used as inputs by EPnP (Section 4.4), the match quality can affect the final navigation accuracy, despite the use of RANSAC. As for time, using the same FLANN index (Subsection 3.6.1), the more features (outputed by the feature detectors, Sections 3.2 and 3.3) for which a match is searched for, the longer the matching process will take.

To improve on both the accuracy and computation time, some methods were introduced, which filter the outputs of one component before providing them as inputs to the next (Sections 3.5 and 3.7). However, it is clear that the complex interactions between components does not allow for an *a priori* choice of the parameters involved.

For these reason, a coarse grid search was performed to find the set of parameters best suited for the algorithm. First, in Section 7.1.1, the parameters that may be used to change the accuracy and computation time of the algorithm were identified. This was done based on a time distribution analysis using both AKAZE and BRISK with dataset 0 (Section 6.2).

Since the parameters that influence the computation time are also the ones that affect the navigation accuracy, this analysis sufficed. During this analysis, a few changes in some algorithms were made, namely the match-downselection algorithm was improved (Section 3.7). An additional method to limit the features before attempting to match them (Section 3.5) was added in the end, which is why the time distribution plots shown do not include this event.

Next, in Subsection 7.1.2, an incremental grid search was performed through the defined parameter space for each of the feature detectors. Upon finding the best parameter set, it was tested using the original real images. A more in-depth characterisation of the accuracy and computation time was derived from the test results.

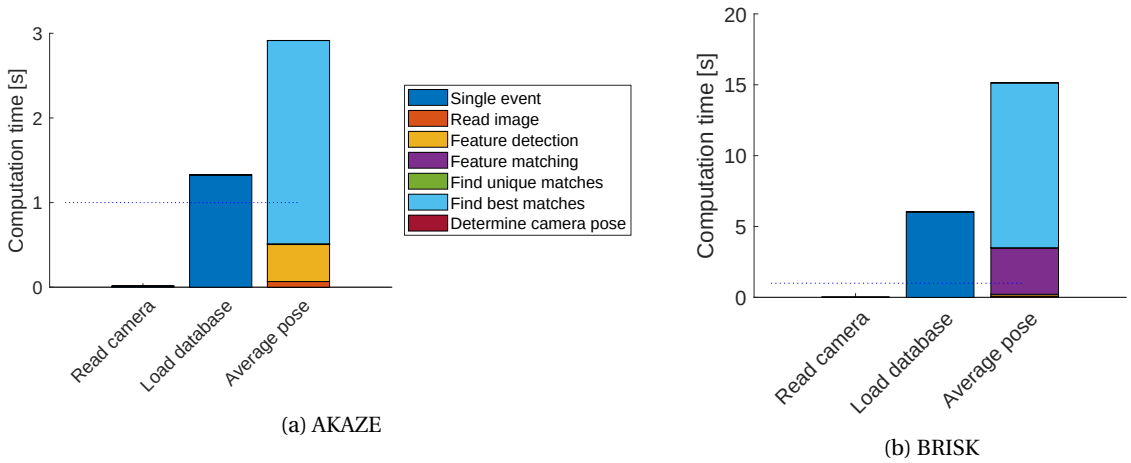
Worthy of note is the precision of the available timing values. These are not the real computation times, since it is not corrected for parallel processes running alongside the navigation software. To determine the consistency of these values, one set of parameters was picked at random and used to generate 10 solutions. The navigation solution was the same in all runs (as expected), but the average time per pose varied about 0.2 s. This difference was enough to make some runs fulfill the requirement while others didn't.

7.1.1. Parameter Identification

As previously mentioned, to identify the significant parameters that influence the computation time and final navigation precision of the algorithm, an analysis was performed using the dataset gathered in TRON, refered to as dataset 0 (described in Section 6.2).

The camera poses were used to get the corresponding rendered images. Since the renderer uses a perfect pinhole camera, the intrinsic parameter matrix, $\mathbf{K}$, was automatically calculated based on the focal length and the camera resolution. The dataset also includes the approximate position of the lamp, which was used to get a better correspondence of the shadows between the real and rendered images. The strength of the lamp was ajusted by trial and error. For the analysis discussed in this section, all rendered images were used to generate the database.

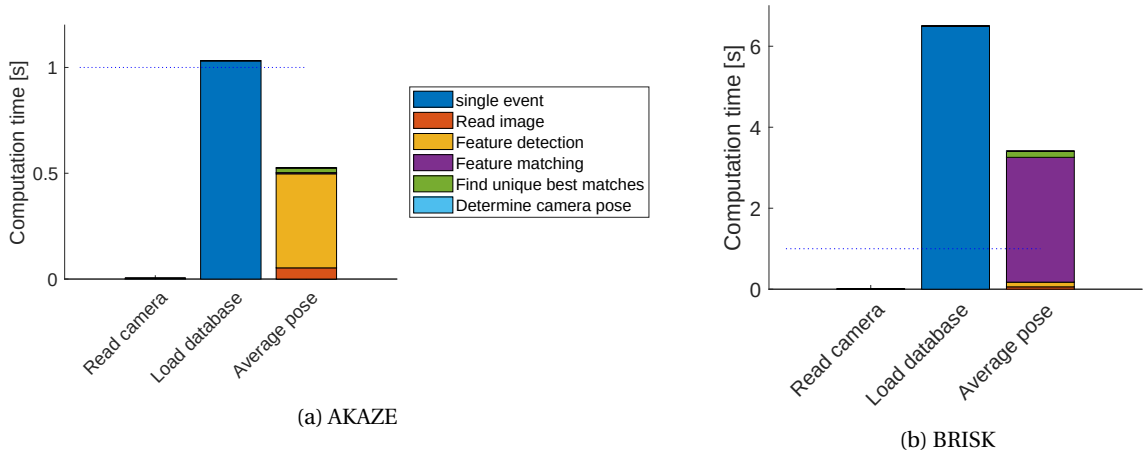
Figure 7.1 shows the computation time, measured in one run, taken by different events of the online script using AKAZE and BRISK, with the first match downselection algorithm (Section 3.7). The events under the



(a) AKAZE

(b) BRISK

Figure 7.1: Time distribution for different events of the online script with first match downselection algorithm



(a) AKAZE

(b) BRISK

Figure 7.2: Time distribution for different events of the online script with improved match downselection algorithm

lable *Single event* are part of the initialisation of the script, thus, they can be performed during a period of the mission in which a timely navigation solution is not yet required. The significant value for the computation time requirement is the *Average pose* total, further referred to as pose time.

Reading the camera parameters file (*Read camera*) takes a negligible ammount of time and can be replaced by hard-coding the required parameters, for each specific mission. *Feature detection* depends only on the descriptor used. As expected, BRISK is faster than AKAZE.

Not depending on any of the algorithm's parameters are *Read camera* and *Read image*. Loading the database is clearly dependent on its size, namely the number of data feature points and the dimension of their descriptors. The remaining events depend on at least one of the algorithm's parameters.

The first thing to note from the results is that the pose time is mostly determined by the process of finding the best matches. Since this value is much greater than that observed for the process of finding the unique matches, it is safe to assume that the algorithm used to make the choice is greatly inefficient, which justified changing the initial algorithm. Since the match quality is already checked during the process of finding the unique matches, the two processes were combined, as described in Section 3.7. The new results are shown in Figure 7.2.

As is readily clear, the improved downselection algorithm decreases the computation time significantly, reducing the match downselection time by two orders of magnitude (which is no longer significantly dependent on N). The other time contributions remain similar to the times shown in Figure 7.1.

Table 7.1: Summary of factors that affect the time required for each event of online script

Event	Determining factor	Relative impact
Loading database	Database size	-
Feature detection	Detection parameters	Low to none
Feature matching	Database size Number query points Index target precision	Medium High Low to none
Finding unique best matches	Database size Number query points	- -
Determining camera pose	Use RANSAC Number best matches (N)	Low to none Medium

The new pose time has different determining events for each of the descriptors. For AKAZE, the most time consuming event is the feature detection. It is possible to alter the parameters of the detection engine used, *e.g.*, the descriptor size, the number of octaves and the threshold. The impact of these changes to the pose time and navigation accuracy requires investigation. Regardless, the pose time achieved already satisfies requirement **SR_3c**.

As for BRISK, the determining event is the feature matching. It has been noted that BRISK detects far more features than AKAZE in the data images. First, this leads to a bigger database. The one used for the study in this section had about 15,000,000 points for AKAZE and almost 87,000,000 for BRISK (around 5.7 times more points than for AKAZE). Second, there will also be more query points for each online image. Third, the BRISK descriptor is bigger than the AKAZE one (512 entries versus 488).

The dimension of the BRISK descriptor is fixed. However, it is possible to change, among other parameters, the threshold score and the number of octaves. A higher threshold can reduce the number of detected features. How this affects the feature detection time and navigation accuracy (if at all) is not immediately clear. Another alternative to reduce the number of points, is to limit the number of data and query features to a fixed number. This alternative may be more robust to illumination changes.

The size of the database can also be reduced by decreasing the number of data images. One final way to change the databases is changing the parameters used to optimise the index, specifically the target precision. So far, this precision has been set to 90%. This high value may lead to slower searches and may be unjustified, thanks to the match downselection and the use of RANSAC in the pose determination.

Finally, it is worthy to note that the camera determination is the least time consuming event. Therefore, it is entirely possible that reducing the number of points used may actually increase the pose time. Since the RANSAC scheme is used, it is also possible that outliers are successfully removed without requiring prior downselection.

Table 7.1 summarises the factors that determine the time required by each relevant event. The database size is dependent on the number of data features. Both the number of data features and of query points can be manipulated through

- Detection parameters (of AKAZE and BRISK)
- Maximum number of feature points

Finaly, the number of data images can also affect the number of data features.

Table **??** also provides a subjective measurement of the relative impact of each factor in the average pose time. The number of query points was found to be the most significant factor, followed by the database size.

The navigation accuracy is dependent on these same parameters. Aditionally, the images and features chosen will also play a role. Choosing five images in the beginning or equally spread through the trajectory will impact the precision significantly whereas it will not lead to a direct change in the computation time (although, indirectly, a larger number of data features are expected from images closer to the target). Likewise, choosing 50 features in the top right corner instead of spread through the image may have a significant impact on the navigation accuracy, while keeping the same computation time.

The detection parameters that impact the number of features detected are

- threshold: lower limit imposed to a point's response value to detect it as a feature (default values: AKAZE - 0.001, BRISK - 30)
- octaves: number of octaves used to build the scale-space (default values: AKAZE - 4, BRISK - 3)
- sub-levels: number of AKAZE sub-levels (in between the octaves; default value: 4)

Table 7.2: Overview of the tuning parameters and their grid points

Parameter	Meaning	Grid points
spacing	Spacing between data images	1, 2, 4, 6, 10
n data features	Limit for data features per image (0 = no limit)	0, 100, 500, 1000, 10000
image division	Number of rows and columns to divide image for feature distribution (Section 3.5)	1, 2, 3, 4
percentage non div	Percentage of features allocated to be found in the overall image (Section 3.5)	0, 0.5
n query features	Limit for query features per image	0, 100, 500, 1000, 10000
precision	Index target precision	0.6, 0.9
use RANSAC	RANSAC switch (1 = use)	0 ,1
n best matches	Number of best matches used for pose determination	0, 10, 20, 50, 100
AKAZE detector		
threshold data	Database threshold	0.0005, 0.001
threshold query	Online threshold	0.001, 0.002
octave	Database octaves	4, 6
descriptor size	Number of bits used for descriptor (0 = full size)	0, 64, 256
BRISK detector		
threshold data	Database threshold	20, 30
threshold query	Online threshold	30, 50
octave	Database octaves	3, 6

Apart from these, AKAZE's descriptor size (default: full size) may also alter the navigation accuracy. Whether any of these parameters affect the feature detection time is unclear: depending on the algorithm used, a higher number of detected features and a more complex descriptor may not require a significantly higher computation time.

After a brief analysis of the impact of the detection parameters in the features found, the effect of sublevels (in AKAZE) was considered negligible. Changes in the number of octaves has a higher but still small effect, although increasing it can lead to larger image structures to be detected. The threshold had the biggest impact, but high deviations from the default values lead to undesirable results.

From the preliminary analysis here discussed, the grid parameters and values have been defined and are shown in Table 7.2. All combinations were considered except for $par_6 = 1$ (*image division*), for which *percentage non div* (which has no effect in this case) was set to 0.

7.1.2. Grid Definition

The grid defined in Table 7.2 contains a total of 560,000 points (420,000 for AKAZE and 140,000 for BRISK). If each case were to take 1 minute to complete (an average below 1 s per pose), the complete search would take about 390 days. This estimate, of course, does not account for the time require to generate the databases.

Since this is not a reasonable time period, the search was reduced incrementally. First, it began using all the points defined. Once a sufficient number of cases (above 400) had been completed, an analysis of the accuracy in terms of the norm of the position error was executed, based on the requirements **SR_3a** and **SR_3b**. The analysis was then used to reduce the number of possible values in a given parameter, and the search continued. The two detectors were analysed separately, since they do not necessarily share the same dependency on the parameters.

The order in which the parameters were changed is shown in Table 7.3.

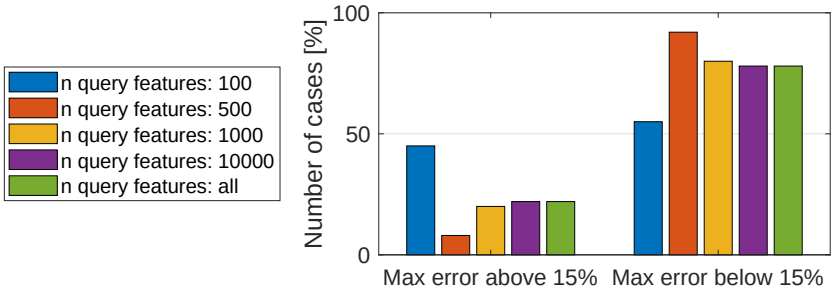
The first change made was removing the possibility of not using RANSAC in the pose determination. This choice was made after only a few cases being solved, since none of the cases without RANSAC had a solution. Next, the grid search process is discussed for AKAZE and BRISK in Subsections 7.1.3 and 7.1.4, respectively.

7.1.3. AKAZE Grid Search

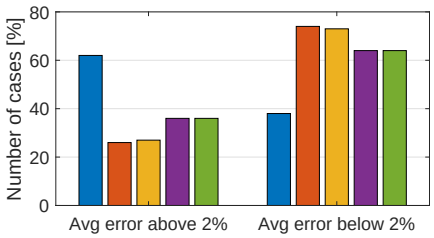
Figure 7.3 shows the histograms used in the first analysis. These histograms show the percentage of cases where the maximum error found was above 15% of the LOS distance (Figure 7.3a), where the average error was below 2% and below 1% (Figures 7.4b and 7.3c, respectively).

Table 7.3: Ordered parameters and respective values for AKAZE and BRISK

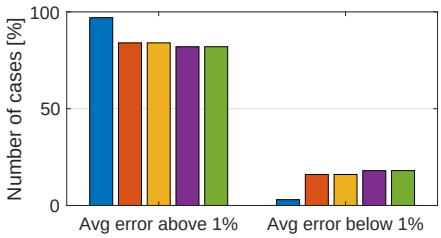
Id	Name	Value		Id	Name	Value	
		AKAZE	BRISK			AKAZE	BRISK
1	*threshold data*	0.0005, 0.001	20, 30	7	*percentage non div*	0, 0.5	
2	*octave*	4, 6		8	*precision*	0.6, 0.9	
3	*descriptor size*	0, 64, 256	(-)	9	*n query features*	0, 100, 500, 1000, 10000	
4	*spacing*	1, 2, 4, 6, 10		10	*threshold query*	0.001, 0.002	30, 50
5	*n data features*	0, 100, 500, 1000, 10000		11	*n best matches*	0, 10, 20, 50, 100	
6	*image division*	1, 2, 3, 4		12	*use RANSAC*	0, 1	



(a) Maximum error, 15%



(b) Average error, 2%



(c) Average error, 1%

Figure 7.3: Case distribution based on norm of position error according to *n query features* (set 0)

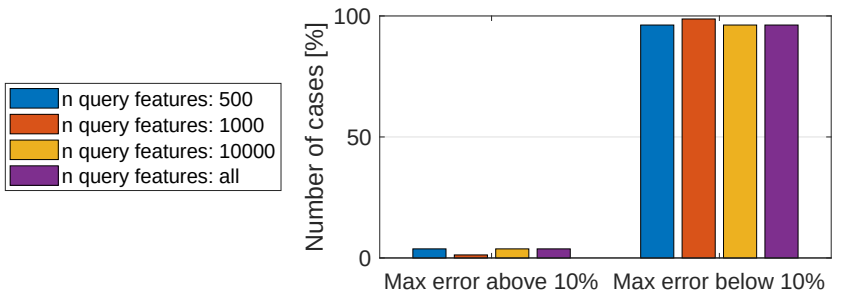
For example, in Figure 7.3a, from all the navigation solutions (each containing 61 camera poses) obtained with the parameter *n query feature* set to 100, about 50% of these solutions had a maximum error above 15%. In other words, at least one of the poses in 50% of these navigation solutions had a position error with respect to the real pose higher than 15% of the LOS distance at that point. In Figure 7.3c, almost 100% of these navigation solutions had an average error (the average of the error observed in all the 61 poses calculated) above 1% of the LOS distance.

The values used for the histograms were lowered as the quality of the solutions improved throughout the search. For example, in set 11, the values used were 10%, 1% and 0.7%, respectively (see Figure 7.4).

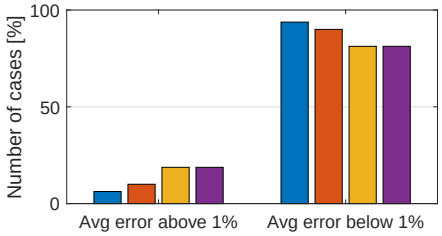
For set 0, a total of 500 cases (100 for each of the values under inspection) were used. These consisted of all the results under *image division* of 1-3. In other words, the cases considered were obtained with all parameters from 1 (*threshold data*) to 5 (*n data features*) equal to their first possible value; with *image division* equal to 1, 2 or 3; and with all the variations of the remaining parameters still under study (the value 0 for parameter 12 has already been rejected).

From the resulting plots, it was observed that the cases for which 100 query features were used had significantly lower performance than the remaining ones. In Figure 7.3a only about half of the cases have a maximum error below 15% of the line of sight distance compared to about 75% for the other values. In Figure 7.3c, almost none of the cases with this value satisfy requirement **SR3_a** (average error below 1%). From this analysis, it was decided to reject the value 100 for the parameter *n query features*.

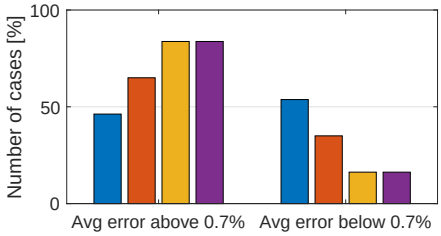
To note, this procedure does not guarantee that the global optimum will be found. In fact, this is not the objective, since doing so would only optimise the algorithm for the particular case in study. Instead, this method incrementally reduces the search to the most consistently performing parameters. The hidden assumption is



(a) Maximum error, 10%



(b) Average error, 1%



(c) Average error, 0.7%

Figure 7.4: Case distribution based on norm of position error according to *n query features* (set 11)

that the impact of a given parameter is relatively consistent regardless of the remaining parameters. In other words, if a parameter has a significant impact on the performance of the algorithm, keeping it constant will yield similar accuracy for any combination of the remaining ones. If this is not the case (the accuracy varies greatly while keeping a given parameter constant), then there is at least one other parameter more significant to the results.

In the example of Figure 7.3, there are some cases for which the average error is below 1% of the line of sight distance when using 100 query features. However, the plots indicate that this value does not tend to lead to good performances. It is possible that these few cases are benefiting from the effect of a different parameter, which potentially is also responsible for the cases of average below 1% with different number of query features.

In practice, for a given set of cases being considered, plots were made for different parameters until one of high significance was found. Once a value is rejected, it is no longer used to get new solutions. During the search an improvement of the overall performance was clearly visible.

Apart from rejecting values of highly significant parameters, in some cases, to increase the speed of the search, some values were rejected for parameters which did not seem to contribute to the performance. This was only done after consistently observing nearly identicle results through multiple sets along the reduction process.

An example is shown in Figure 7.4. Once again, the number of query features is under study. Here it is noted that the results using either all or 10,000 query features are exactly the same. The same is visible in Figure 7.3 and was observed through the intermediate sets. This can be due to, for example, detecting less than 10,000 query features per image, which would mean the same features were being used in both cases. To reduce the search time, the value 0 (use all query features) was rejected. Values rejected in this manner can later be readressed, if necessary.

Table 7.4: Grid search process for AKAZE

Id	Cases considered	Number of cases	Parameter in study	Observations	Decision
0	$par_6 \in \{1,2,3\}$	500 cases; 100 each	par_9 – *n query features*	$par_9 = 100$, 50% cases with maximum error over 15% (compared to 25%); nearly 100% cases with average error above 1%	Reject $par_9 = 100$
1	$par_6 \in \{1,2,3\}$	400 cases; 80 each	par_{11} – *n best matches*	$par_{11} = 0$ (all), over 75% with maximum error above 15% (compared to near 0%)	Reject $par_{11} = 0$
2	$par_5 = 0$	448 cases; 112 each	par_{11} – *n best matches*	$par_{11} \in \{10,20\}$, 100% with average error above 1 %	Reject $par_{11} = 10,20$
3	$par_5 \neq 10000 \wedge par_7 = 0$	512 cases; 128 each	par_6 – *image division*	$par_6 = 1$, less than 20% with average error below 1% (compared to over 40%)	Reject $par_6 = 1$
4	$par_5 \neq 10000$	768 cases; 384 each	par_7 – *percentage non div*	$par_7 = 0.5$, 15% less cases with average below 1%	Reject $par_7 = 0.5$
5	$par_5 \neq 10000$	384 cases; 192 each	par_{11} – *n best matches*	$par_{11} = 50$, 25% less cases with average below 1%	Reject $par_{11} = 50$
6	$par_5 \neq 10000$	192 cases; 96 each	par_8 – *precision*	Very similar results (also seen in previous sets)	Keep $par_8 = 0.6$
7	$par_3 = 0$	600 cases; 120 each	par_5 – *n data features*	$par_5 = 100$, near 80% cases with average above 2%; only 20% cases below 1% (compared to over 60%); only value with maximum error above 15%	Reject $par_5 = 100$
8	$par_3 = 0$	480 cases; 240 each	par_{10} – *threshold query*	$par_{10} = 0.002$, 25% less cases with average below 1%	Reject $par_{10} = 0.002$
9	$par_2 = 4$	720 cases; 240 each	par_3 – *descriptor size*	$par_3 = 64$, less than 25% cases with average below 1% (compared to 75%); only value with maximum above 15% and average above 2%	Reject $par_3 = 64$
10	$par_2 = 4$	480 cases; 160 each	par_6 – *image division*	$par_6 = 4$, 25% less cases with average below 1%	Reject $par_6 = 4$
11	$par_2 = 4$	320 cases; 80 each	par_9 – *n query features*	Exact same results for $par_9 = 10000$ and $par_9 = 0$ (also seen in previous sets)	Keep $par_9 = 10000$
12	all	960 cases; 480 each	par_1 – *threshold data*	$par_1 = 0.0005$, around 30% cases with average below 1% (compared to around 90%); only value with average above 2%	Reject $par_1 = 0.0005$
13	all	480 cases; 120 each	par_5 – *n data features*	$par_5 = 500$, 100% cases with average below 1% and maximum below 10%; over 60% cases with average below 0.7% (compared to 30%)	Chosen $par_5 = 500$
14	all	120 cases; 40 each	par_9 – *n query features*	$par_9 = 500$, near 50% cases with average below 0.6% (compared to 15%)	Chosen $par_9 = 500$
15	all	40 cases; 20 each	par_6 – *image division*	$par_6 = 3$, 100% cases with average below 0.7% and maximum below 7%; only value with average below 0.6%	Chosen $par_6 = 3$

Table 7.5: Tuned parameters for AKAZE

Id	Name	Value	Id	Name	Value
1	*threshold data*	0.0005	7	*percentage non div*	0
2	*octave*	4/6	8	*precision*	[0.6]
3	*descriptor size*	0	9	*n query features*	500
4	*spacing*	6	10	*threshold query*	0.001
5	*n data features*	500	11	*n best matches*	100
6	*image division*	3	12	*use RANSAC*	1

Table 7.4 summarises the reduction process. The first column shows the index used to identify the set described in the corresponding row. Under cases considered, the subset of parameters used for the plots is given. For example, for set 7, the histograms plotted included the runs for which $par_1 = 0.0005$, $par_2 = 4$ and $par_3 = 0$, and excluding all the parameter values rejected in the previous sets. In other words, including all the combinations possible with the reffered parameter fixed to the indicated value and the previous parameters (with smaller indeces) fixed to the first value of the grid (seen in Table 7.3).

The parameter in study reffers to the parameter used to divide the cases for the histograms shown. The conclusions taken from them (and expressed under "Observations") will reflect in a decision on values of this parameter.

In set 3, the parameter *image division* is being studied. Since, when this parameter has the value 1 (the whole image is considered as a whole, see Section 3.5) the parameter $percentagenondiv$ has no effect, for these runs, it was always 0. To allow a valid comparison between the different values of *image division*, only the runs for which $percentagenondiv$ was set to 0 were considered.

Having completed the procedure described above, 20 cases remained. Inspecting their maximum and average norm of the position error, the best result (in both measures) was found for two different cases (differing in the number of octaves). The average and maximum errors were of 0.4788% and 1.0618% of the line of sight distance, respectively. The parameter values of these two cases are summarised in Table 7.5.

At this stage, the impact of the FLANN index precision (par_8) on the accuracy is still unknown. Between the two different values for the number of octaves, $par_2 = 4$ is chosen at this stage, being the default value and potentially slightly decreasing the time required to generate the database.

These parameters were used to validate the algorithm (Section 6.8) with the original images (before undistortion).

7.1.4. BRISK Grid Search

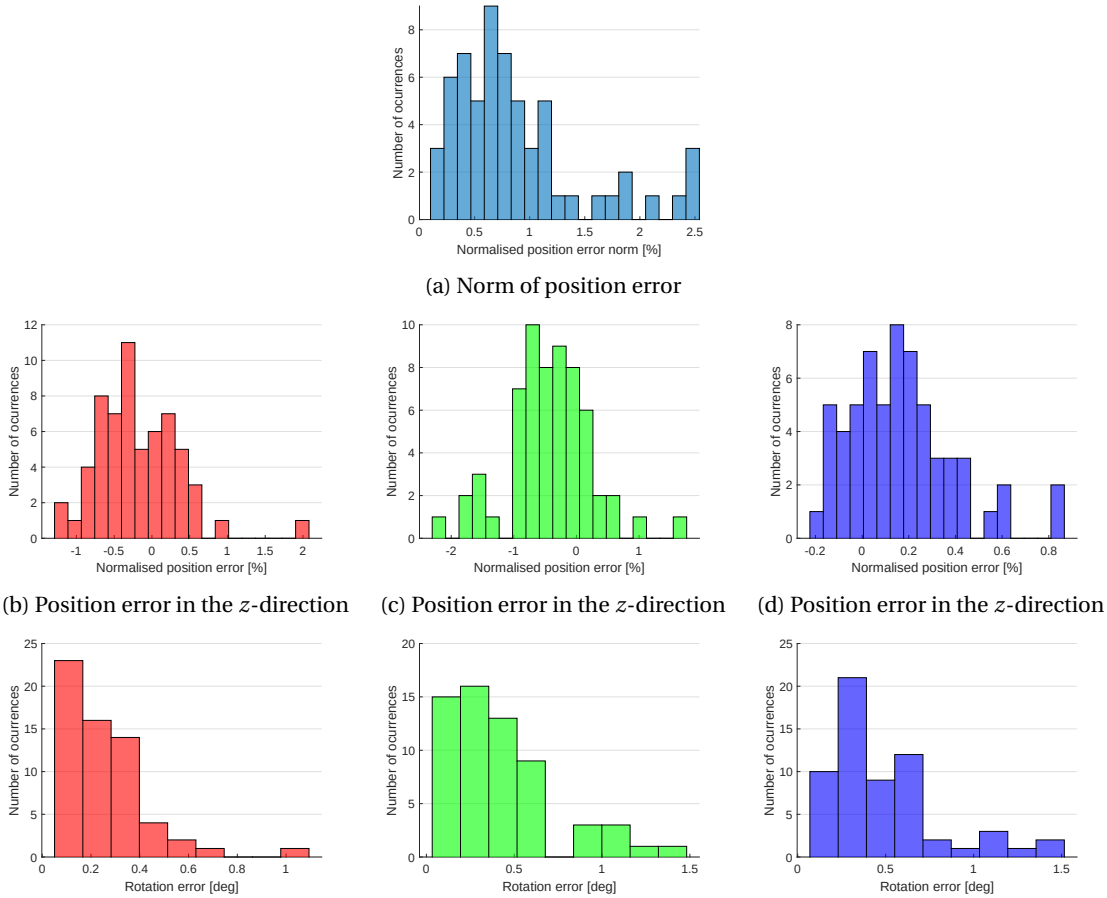
The grid search for BRISK was reduced using the same method as for AKAZE. Table 7.8 summarises the decisions made in this phase. It is to note that the performances did not improve as much as for AKAZE. The restrictions in place are shown in Table 7.6.

From the resulting cases, the one for which the average error was minimum had a maximum error of 4.7% of the LOS distance. To determine whether this was an outlier, the associated parameters (restrictions shown in Table 7.6, *octave* = 6, *spacing* = 1, *n query features* = 1000, *n best matches* = 100) were used to derive the navigation solution for the original images. The result included three measurements with an error above 3%. In a total of 61 poses, this does not meet requirement **SR_3b**.

In an effort to find better performances, the search was expanded to include *precision* of 0.9 and *threshold query* of 30 (two values removed from the search to increase its speed, since no significant relation had been

Table 7.6: Parameter values after first grid search reduction for BRISK

Id	Name	Value	Id	Name	Value
1	*threshold data*	30	7	*percentage non div*	0
2	*octave*	-	8	*precision*	[0.6]
3	*descriptor size*	-	9	*n query features*	500,1000
4	*spacing*	-	10	*threshold query*	50
5	*n data features*	1000	11	*n best matches*	50,100
6	*image division*	1	12	*use RANSAC*	1



(a) Norm of position error

(b) Position error in the z-direction

(c) Position error in the z-direction

(d) Position error in the z-direction

(e) Orientation error of the x-axis

(f) Orientation error of the y-axis

(g) Orientation error of the z-axis

Figure 7.5: Error histograms for navigation solutions using the tuned BRISK parameters on the original images from dataset 0

found), as well as *n data features* and *n query features* of 2000 and 5000. The complete set of results was then filtered for the cases with an average error below 1%, a maximum error below 3% and a pose time below 2 s. From a total of 4570 cases, 16 were found to meet these conditions, only one of which had been removed by the analysis (with *n query features*= 500).

These cases were reduced to 10, by ignoring the ones for which all performance metrics were above the ones found for another set of parameters. Each resulting set was used to compute the navigation solution using the original images. One set yielded no error above 3%. The performance histograms are shown in Figures 7.5 (for accuracy) and 7.6 (for the average time taken from 10 runs of the online script).

The average time found was 1.1 s. Although higher than allowed by requirement **SR_3c**, the difference is smaller than the precision of the measurements (0.2 s). Furthermore, the optimisation of the algorithm could reduce this time. The average error norm obtained was 0.88%. The angle errors were of the same magnitude for both the y- and z-axis and lower for the x-axis. Similarly, the error was lower in the z-direction and of the same magnitude for the remaining ones. The same was observed, for the position errors, with AKAZE in Section 6.8.

The parameters used are given in Table 7.7. During the expanded search, these parameters yielded the second lowest average error (0.71%) and the lowest maximum error (1.79%). However, it is important to note that the performance achieved with the described restrictions was not as robust as observed for AKAZE. During the selection process for the feature detectors, BRISK was identified as being a faster but less precise detector in comparison to AKAZE, which could explain the higher variability here observed. As such, it is possible that the chosen parameters may not perform as well for a different trajectory.

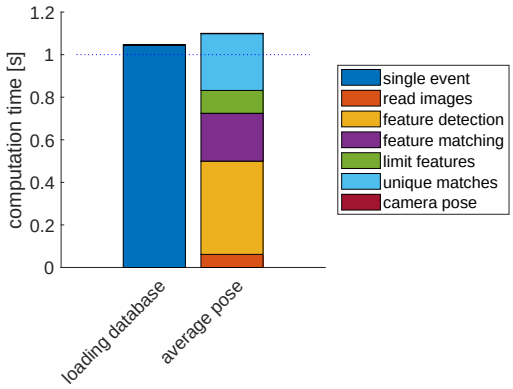


Figure 7.6: Average time distribution for 10 runs of online script using the tuned BRISK parameters on the original images from dataset 0

Table 7.7: Tuned parameters for BRISK

Id	Name	Value	Id	Name	Value
1	*threshold data*	30	7	*percentage non div*	0
2	*octave*	3	8	*precision*	0.9
3	*descriptor size*	-	9	*n query features*	5000
4	*spacing*	2	10	*threshold query*	30
5	*n data features*	1000	11	*n best matches*	100
6	*image division*	1	12	*use RANSAC*	1

Table 7.8: Grid search process for BRISK

Id	Cases considered	Number of cases	Parameter in study	Observations	Decision
0	$par_5 = 0$	560 cases; 112 each	par_9 – *n query features*	par_9 = 100, none with average below below 2% (compared to 75%); 10% with maximum error below 15% (compared to 75%)	Reject $par_9 = 100$
1	$par_5 = 0$	448 cases; 112 each	par_{11} – *n best matches*	$par_{11} = 0$ (all), none within requirements; $par_{11} = 20$, no case with average below 1% (compared to about 10%)	Reject $par_{11} \in \{0, 20\}$
2	$par_5 \in \{0, 500\}$	448 cases; 224 each	par_8 – *precision*	Very similar results	Keep $par_8 = 0.6$
3	$par_4 = 1$	560 cases; 112 each	par_5 – *n data features*	$par_5 = 100$, none with average below 1%; none with maximum error below 10%	Reject $par_5 = 100$
4	$par_4 = 1$	448 cases; 112 each	par_9 – *n query features*	$par_9 \in \{0, 10000\}$, near 100% with time per pose above 1.2 s; no significant improvement in average	Reject $par_9 \in \{0, 10000\}$
5	$par_4 \in \{1, 2, 3\}$	672 cases; 336 each	par_{10} – *threshold query*	Similar average error; $par_{10} = 50$, significant faster (about 0.4 s); 10% more with maximum error below 10%	Keep $par_{10} = 50$
6	$par_2 = 3$	560 cases; 140 each	par_5 – *n data features*	Very similar results for $par_5 \in \{0, 10000\}$; $par_5 = 500$, 40% with average below 2% (compared to over 75%); 20% with maximum error below 10% (compared to over 50%)	Reject $par_5 = 500$ Keep $par_5 = 10000$ (over $par_5 = 0$)
7	all	1120 cases; 560 each	par_1 – *threshold data*	$par_1 = 50$ near 0% within requirements	Reject $par_1 = 50$
8	$par_7 = 0$	320 cases; 80 each	par_6 – *image division*	$par_7 = 1$, around 10% with average error below 1% (compared to near 0%); 25% more with maximum error below 10%	Choose $par_7 = 1$
9	all	80 cases; 40 each	par_5 – *n data features*	$par_5 = 1000$, only with average error below 1%	Choose $par_5 = 1000$

7.2. Simulated Missions

To determine the performance of the algorithm using a trajectory as close to realistic as allowed by the laboratory setup (see Chapter 5), a trajectory was generated using a trajectory optimiser (see Subsection 6.4.3) and adapted to the requirements and limitations of the TRON laboratory (see Appendix B). To analyse its robustness to the disturbances expected in a landing scenario, the same optimiser was used to generate disturbed trajectories by changing the initial state of the lander, namely its position and velocity. For this analysis, the same database generated with the nominal trajectory was used.

The trajectories used and the two datasets generated with it are described in Subsection 7.2.1. The results for the nominal trajectory and for the disturbed trajectories are given in Subsections 7.2.2 and 7.2.3, respectively.

7.2.1. Datasets I and II

For the final performance and sensitivity analysis with ideal data, a new set of trajectories was used. These trajectories cover a much larger distance than the one used to test and validate the algorithm (see Section 6.8) and the images are sparser. It also includes some variation in the camera orientation. The dataset include the associated camera calibration results, the approximate position of the lamp, the position of the terrain reflectors and various sub-sets of online images and their state measurements.

Due to limitations associated to the LT, the instrument had to be placed in three different positions. Thus, the trajectories were sectioned accordingly and three measurements of the reflectors' positions were made. Since the lamp was not moved between measurements of different sections, and it is only possible to derive an approximate position, its position was only measured for the first LT position.

The trajectories were generated by the same optimiser used for the ATON project (briefly described in Subsection 6.4.3). The nominal trajectory had an initial position equal to $\begin{pmatrix}-101 & 50 & 1500\end{pmatrix}^T$ and initial velocity equal to $\begin{pmatrix}10 & -10 & -40\end{pmatrix}^T$ in the reference system centred in the landing target, where the Z-axis is perpendicular to the landing surface. It was used to generate the database and to determine the performance of the software for the ideal case.

The disturbed trajectories were generated by changing the initial conditions in the optimiser, namely with random uniform error distributions of 100 m of radius from the initial state and 10 m/s for the velocity magnitude. These trajectories were used to analyse the robustness of the navigation system to possible deviations in the context of a mission.

A scale of 1:100 was applied. The process used to transform these trajectories to KUKA commands and the restrictions associated are discussed in Appendix B.

Using these trajectories, two datasets were obtained. Dataset I, includes all trajectory points that can be measured with the LT positions used (see Appendix B for details) and the corresponding online images. The trajectories are represented in Figure 7.7a. The nominal trajectory is represented in blue, with the points used represented by circles. The disturbed trajectories are represented in red, with the points used marked by crosses.

It also includes, as with all datasets, the camera calibration file and the lamp position. Finally, it contains the reflectors' measurements for each of the three LT positions (one for each section).

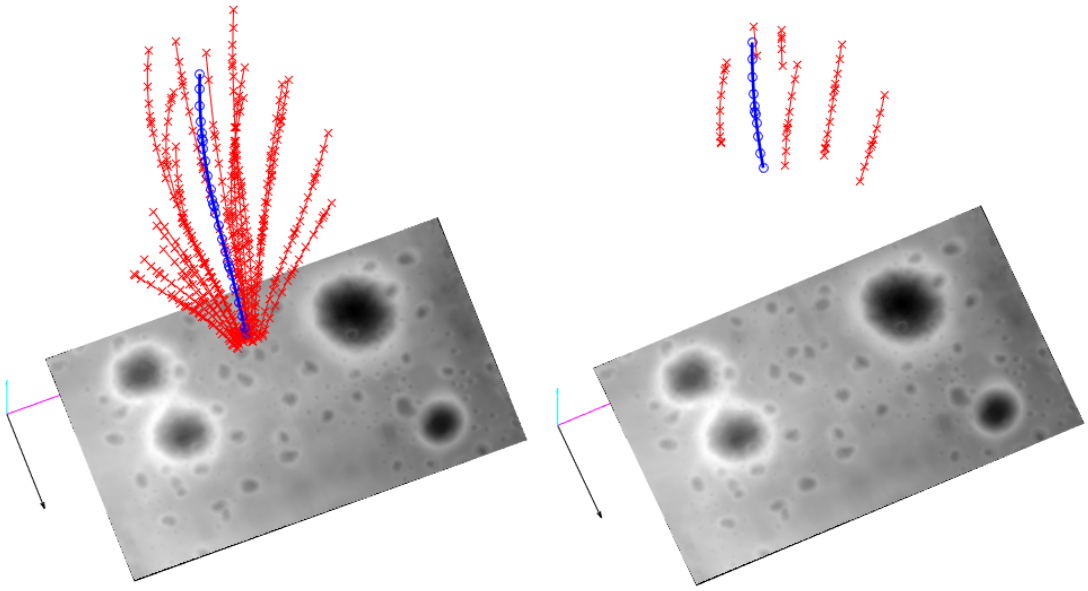
After analysing the results obtained using dataset I, a new effect was recognised to have an impact on the precision of the navigation solution (as discussed in Subsection 7.2.2). To determine whether this effect was indeed the determining factor of the resulting navigation precision, the trajectory points belonging to the section furthest from the model were rerun in TRON using a different camera lens (with a smaller FOV), resulting in dataset II.

Unfortunately, the lamp was moved between the acquisition of the two datasets; the camera could not be focused as well as for dataset II; and the time constraints limited the number of points considered. These points are shown in Figure 7.7b.

7.2.2. Nominal Trajectory

The first results obtained from the new dataset (dataset I) made use of the tuned parameters as obtained from the grid search performed on dataset 0 (see Chapter 7.1). These parameters are summarised in Tables 7.5 and 7.7, for AKAZE and BRISK respectively.

Since the number of images and spacial distance between them differs significantly between the two datasets, parameter 6 – *spacing*, which controls the number of images used in the database – was set to 1 (instead of 6 in AKAZE and 2 in BRISK). This means all rendered images were used for the databases.



(a) Points used for dataset I

(b) Points used for dataset II

Figure 7.7: Trajectory points and camera axis used to capture the images

The results are represented in Figures 7.8 and 7.9 for AKAZE and BRISK, respectively, with the same format used in Chapter 7.1. Since the angle and position errors are correlated, the histograms for the angle errors were ommited, given their lesser relevance in assessing the fulfillment of the requirements.

The first thing worthy of note is that the errors are significantly higher than the ones obtained with dataset 0. Two reasons could be used to explain this disparity.

First, has mentioned above, the images are much sparser than in dataset 0. In this last set, the camera positions are almost equally spaced by 5 cm and the trajectory covers a total of about 3 m with 61 images. The image spacing of 6 used in AKAZE, corresponds to a distance of about 25 cm between camera positions. On the other hand, in dataset I, the distance between camera positions is uneven (due to the method used to choose the nodes for the trajectory optimisation, explained in Subsection 6.4.3), ranging from a minimum of 6 cm to a maximum of 60 cm. This trajectory covers almost 9 m with only 28 images.

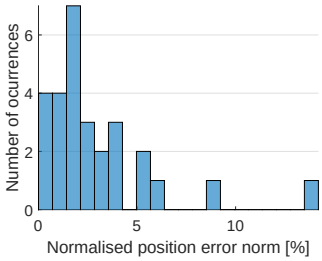
Second, the distance from the model is much larger in dataset I compared to dataset 0. In dataset 0, the first and last camera poses were positioned with $z_M \approx 4$ m and $z_M \approx 1$ m, respectively; whereas for dataset I, the values were $z_M \approx 10$ m and $z_M \approx 1.4$ m.

To check which of these factors is most relevant to the observed error, Figures 7.10a and 7.10b show the position errors obtained with AKAZE as a function of the distance between cameras and the distance to the model, respectively. The distance between points was calculated between the current and previous point in the trajectory. The distance value for the first point was taken to be equal to the second one. From the plots, it is clear that the distance to the model has a major impact in the position error, not visible with the distance between points.

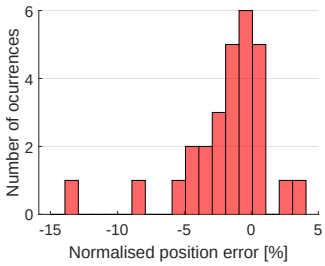
Since the position errors have been normalised with the LOS distance, the effect of lower resolution of the model (at higher distances, each pixel corresponds to a bigger area of the model) is already accounted for. However, due to the limited area of the terrain model, at higher distances the feature detection area is smaller. A small area of the image where features are available leads to bigger orientation errors. Since the position errors are correlated with the orientation errors (remembering that EPnP begins by finding the camera orientation as explained in Section 4.4.1), a smaller feature detection area also leads to higher position errors.

The search area was given as the percentage of white pixels in the mask used for an image. As seen in Figure 7.10c, the feature detection area is a better predictor of the position errors than the distance to the model. Namely, the two points between 2 and 4 m distance to the model with a normalised error around 5% seem to break the tendency shown in Figure 7.10b, but not in Figure 7.10c (where they are at between 40% and 60% of the image area).

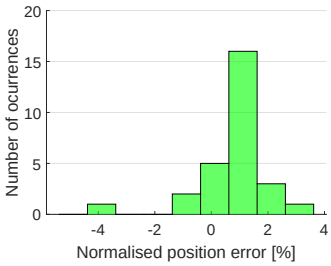
To check this effect, a new dataset was obtained (dataset II, described in Subsection 7.2.1). A new camera



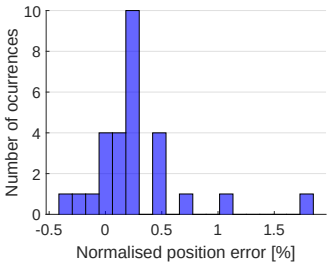
(a) Norm of position error



(b) Position error in the x-direction

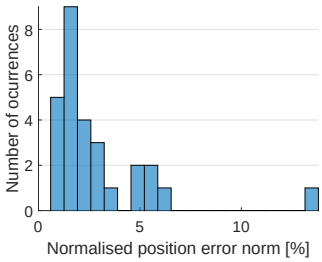


(c) Position error in the y-direction

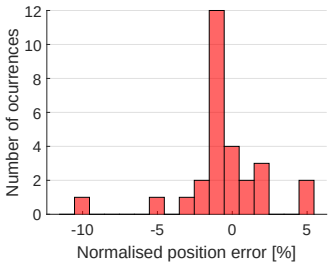


(d) Position error in the z-direction

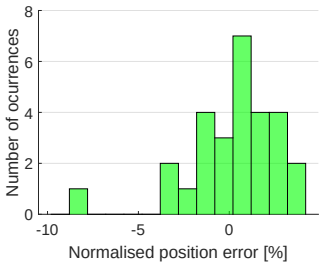
Figure 7.8: Error histograms for navigation solutions using the tuned AKAZE parameters on the original images from dataset I



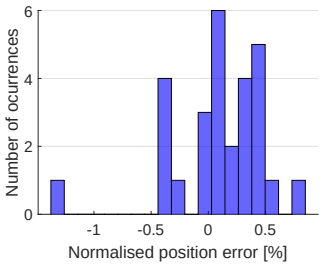
(a) Norm of position error



(b) Position error in the x-direction

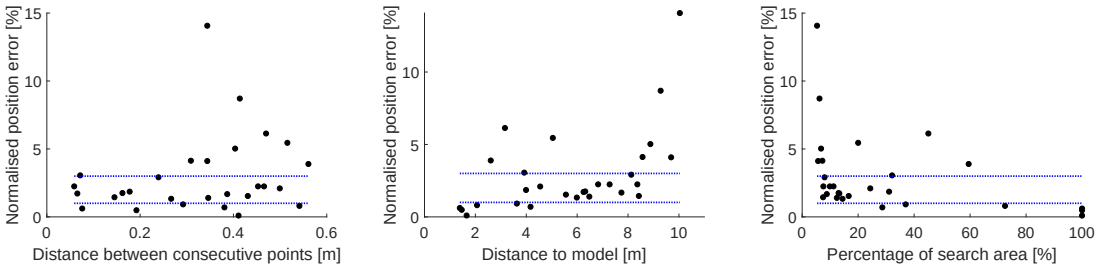


(c) Position error in the y-direction



(d) Position error in the z-direction

Figure 7.9: Error histograms for navigation solutions using the tuned BRISK parameters on the original images from dataset I

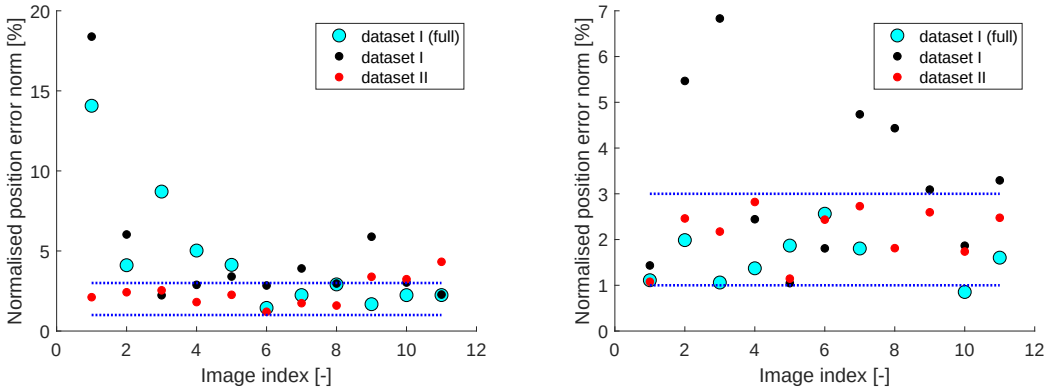


(a) Position errors as a function of distance between camera positions

(b) Position errors as a function of distance to the terrain model

(c) Position errors as a function of the feature detection area

Figure 7.10: Norm of position error and errors in individual axis directions with AKAZE



(a) AKAZE

(b) BRISK

Figure 7.11: Comparison between norm of position error for the different datasets

with a different FOV was used to run the section of the nominal trajectory furthest from the terrain model. Since the FOV is different, although the distance to the model is the same, the area occupied by the model in the image will be different. The camera used had a smaller FOV, thus, for the same distance, the area is bigger. Therefore, smaller errors are expected for the same points.

Due to time constraints, only the furthest part of the trajectory – corresponding to the first 11 positions – was rerun. To guarantee a fair comparison between the two sets, new solutions for dataset I were obtained using only these 11 positions to build the database. A comparison between the three solutions (dataset I with full and reduced database; and dataset II) are compared in Figures 7.11a and 7.11b for AKAZE and BRISK, respectively. In these images, the norm of the position error is ploted as a function of the image number.

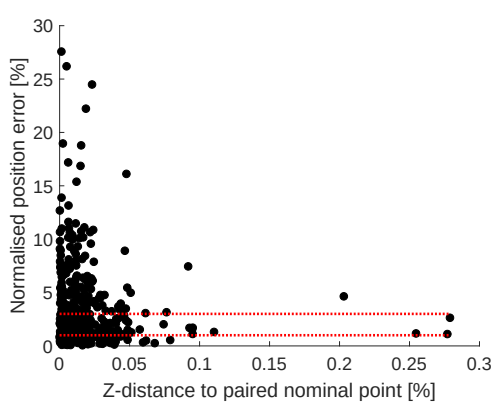
Two things can be noted from these plots. First, for BRISK the navigation precision worsens significantly when less images are used for the database, even when these images are outside the flown trajectory. Second, despite this effect, reducing the FOV of the camera (thus increasing the feature's search area) improved the precision with respect to the dataset I solution for both detectors. With AKAZE, this solution was better than for the full dataset I solution, whereas for BRISK the accuracy was similar in both cases.

7.2.3. Disturbed Trajectories

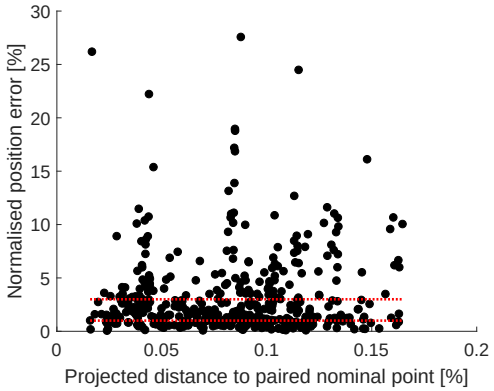
The disturbed trajectories of dataset I were used to test the algorithm's sensitivity to differences in the camera states between the nominal trajectory (used to generate the database) and the real trajectory (which has been disturbed).

To quantify the disturbances, the points of the disturbed trajectory were compared to the nominal trajectory point for which the difference in the z_M-coordinate was smallest. This choice was made considering that the feature's descriptors are affected by zoom but not by translation. Thus it is expected that the biggest number of matches used will be to the database image with the closest z_M-value, which justifies this pairing.

The measures for the disturbances will be the difference of z_M-values between the disturbed and data point

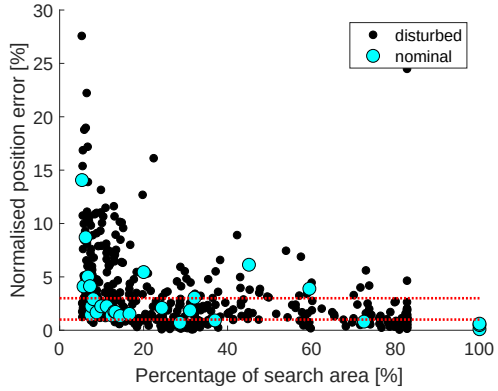


(a) Norm of position error as a function of z-distance

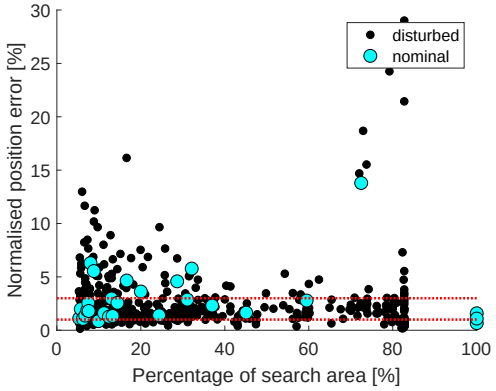


(b) Norm of position error as a function of projected distance

Figure 7.12: Norm of position error as function of distances to paired nominal points normalised with the LOS distance for AKAZE



(a) AKAZE



(b) BRISK

Figure 7.13: Norm of position error as function of the search area

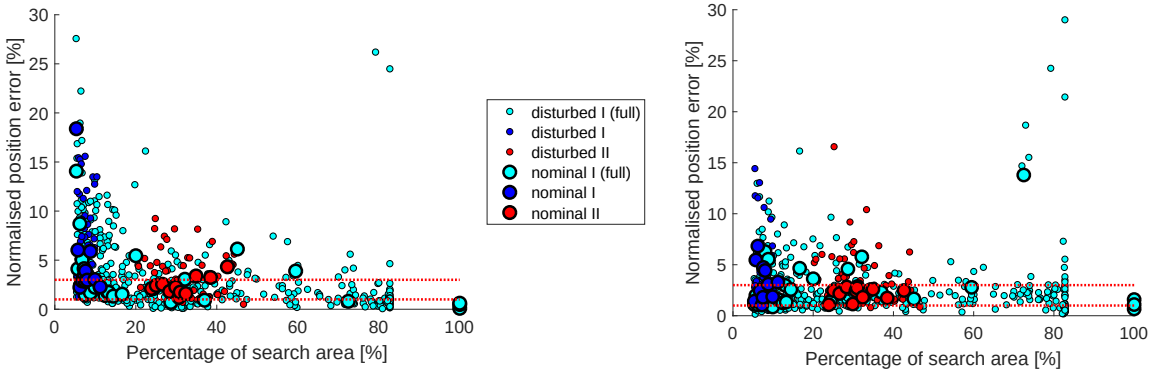
of the pair – z-distance – and the projected distance – ignoring the z_M-dimension – both normalised with the LOS distance (as was done with the position error). Figures 7.12 shows the norm of the position error as a function of these distances for AKAZE, with similar results for BRISK.

The plots show no significant relation between the distances and the position errors. Recalling the effect of the search area on the position error, it is possible that this effect is still the determining factor for the disturbed trajectories. Figure 7.13 shows the norm of the position error obtained for the disturbed trajectories as a function of the search area for AKAZE and BRISK. The same values obtained for the nominal trajectories are overlaid as blue circles, showing that the relation between the two variables is similar for the nominal and disturbed trajectories.

Since it is possible that this effect may be hiding the effect of the trajectory disturbances, some points of the disturbed trajectories were run with a new camera lens with a smaller FOV (dataset II). Figures 7.14a and 7.14b show the norm of the position error as a function of the search area for the disturbed (dots) and nominal (circles) points, for the three datasets (dataset I with full (cyan), and reduced (blue) database; and dataset II (red)), for AKAZE and BRISK, respectively.

The fact that the results of every set fall into the same tendency indicates that the differences in precision observed are due to the available image area for feature detection.

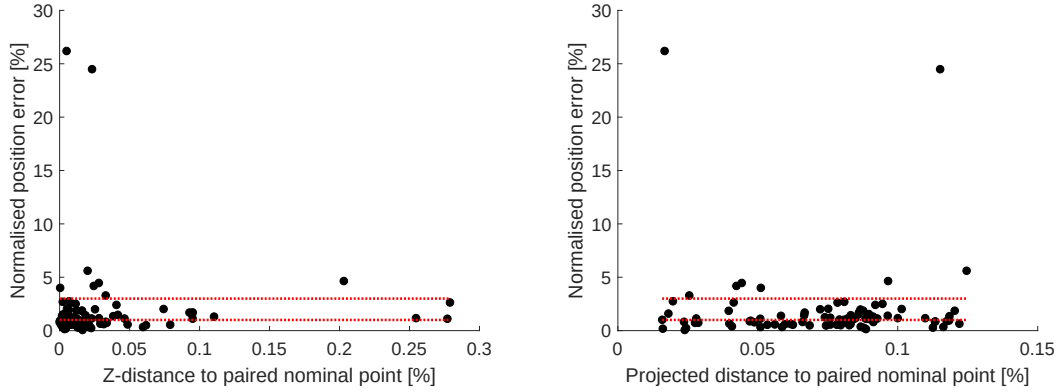
Observing these plots, it is visible that, for all datasets, the nominal solutions tend to be more precise than the disturbed solutions. However, even after reducing the effect of the search area in the navigation error, no clear relation was observed between the distance between a disturbed trajectory point and paired nominal



(a) AKAZE

(b) BRISK

Figure 7.14: Norm of position error as function of the search area for the three datasets



(a) Norm of position error as a function of z-distance

(b) Norm of position error as a function of z-distance

Figure 7.15: Norm of position error as function of distances to paired nominal points for AKAZE and search area above 60%

one (for both the z-distance and the projected distance). Since the search areas for dataset II were still below 50%, the plots were repeated for the dataset I points for which the area was above 60% (a total of 90 points). These plots are shown in Figure 7.15 for AKAZE, with similar results for BRISK. Still, no relation is observed.

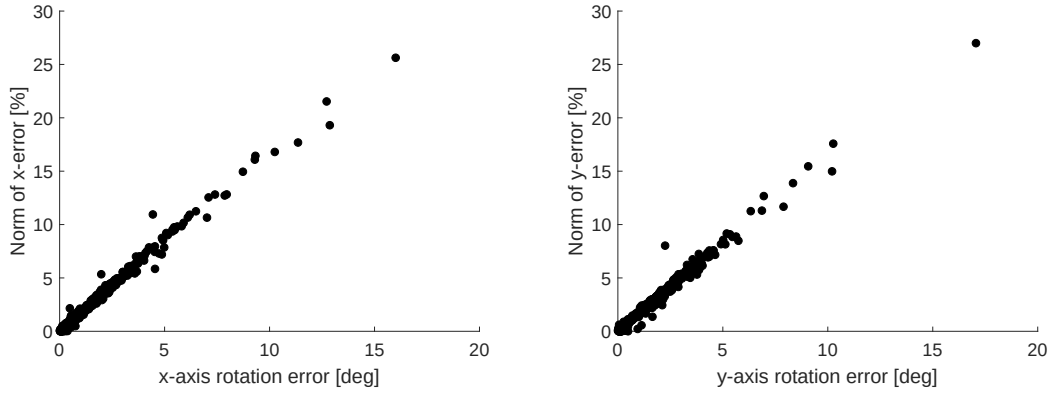
The proposed explanation is that the navigation algorithm is robust to the translation and small orientation changes of the camera expected during a landing mission.

First, the feature detectors used are translation and rotation invariant, meaning that translations of the camera within a plane parallel to the landing surface (terrain model) and orientation changes around the perpendicular axis to it (in this case the Z_M-axis), should not have significant effects on the navigation error, provided enough common area visible to the camera. This translates to the independence with respect to the projected distance.

Second, the detectors also have some zoom invariance. In other words, translating the camera along the axis perpendicular to the landing surface (Z_M-axis) should have only small effects. Thus explaining the independence with respect to the z-distance.

Third, these three effects (image translation, rotation and zoom) were the most prevalent ones, since the orientation changes caused by the expected disturbances were very small. This was further observed in the CE-3 landing images, which, despite including a significant orientation change (during pitch-over), only showed these image transformations.

In sum, the results obtained mostly reveal sensitivity to the available search area. As seen in Figure 7.15, taking a search area above 60% leads to navigation solutions mostly within the requirements specified. For AKAZE, the points outside these bounds are either very close to them or far enough to be detected as outliers by a filter. BRISK shows a much higher dispersion of navigation accuracy for larger search areas.



(a) X_M-direction (b) Y_M-direction

Figure 7.16: Position error as function of orientation error

No relation was found between the navigation solution and the distance to the closest nominal trajectory point. This leads to the hypothesis that, as long as the common area between an online image and the closest database image is above a certain value, the navigation solution will be robust to the trajectory disturbances expected during a lunar landing. Such hypothesis was not tested due to time constraints.

7.2.4. Correlation Between Errors

As previously mentioned, EPnP uses the calculated camera orientation solution to compute its position. Therefore, the two errors are correlated. These correlations were clearly observed from the results obtained and are shown in Figures 7.16a and 7.16b, between the position error and orientation error of the x- and y- directions and axis, respectively.

7.3. Chang'e-3 mission

To test the algorithm with data from a real lunar landing, the CE-3 mission was used as reference (see Section 2.4). In Subsection 7.3.1 the reconstruction of the CE-3 lander's trajectory from the available data is briefly described. The image representability of the images rendered from these points is discussed in Subsection 7.3.2.

7.3.1. Reconstruction of CE-3's trajectory

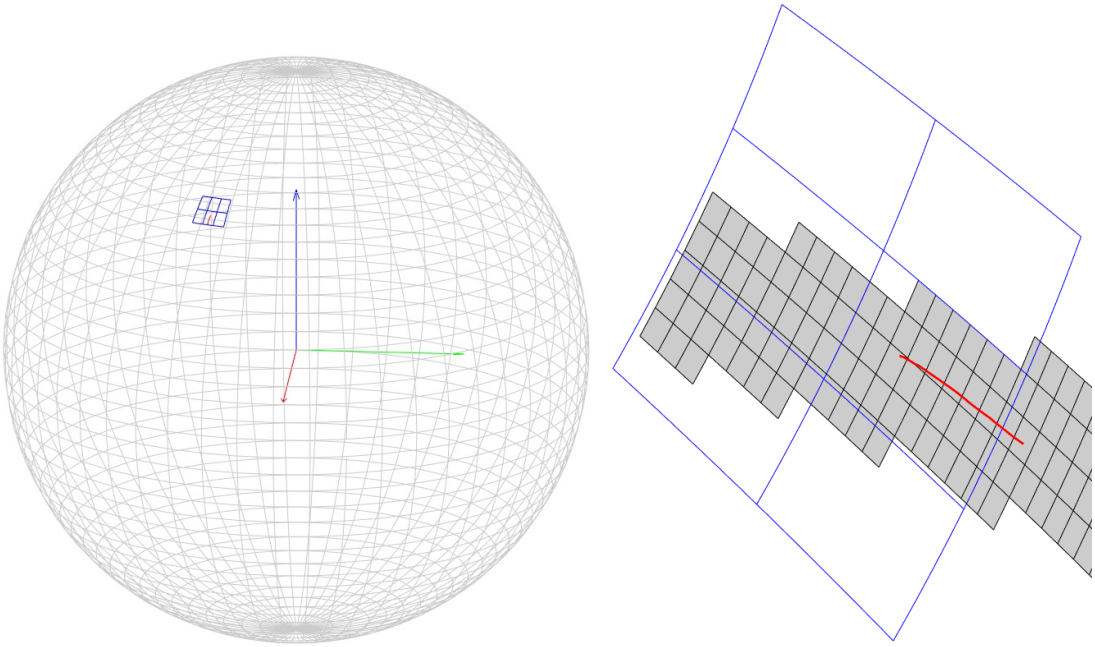
From the plots provided by Liu et al. [2014] (and shown in Figure 2.9), the longitude, latitude and height were obtained by Laura Harrah (a DLR intern prior to the start of this thesis). The radial distance to the lunar centre was determined by adding, to the height, the local surface altitude, obtained from a Kaguya DEM[1].

These data were used to obtain the CE-3 trajectory used in this study. These positions are used as the position of the camera's principal point, which leads to a small position error in the order of metres (dimension of the lander). The orientation of the camera, however, is not known. It was estimated, first based on a roughly determined velocity vector, and then by rotating the camera until the camera view resembled the real landing images. This last step was done through visual inspection of the rendered images obtained.

The first estimate of the orientation at point i of the trajectory was made by aligning the Z_C-axis (see Subsection **??**) in the direction connecting it to point $i+1$ (considered the velocity vector) and the Y_C-axis towards the centre of the Moon and perpendicular to Z_C-axis. For the last point, the Z_C-axis points in the same direction as for the point preceeding it. Next, the frame is rotated around the X_C-axis such that the camera view is similar to the descent images (via visual inspection).

The final trajectory is shown (in red) in Figure 7.17. Figure 7.17a shows the position of the trajectory in the lunar reference sphere. The frame represented is the MCMF frame, $\mathcal{F}_{MCMF}$. Figure 7.17b shows a close-up of the relevant area of the surface. In blue, the positions of the six Kaguya DEM tiles used for the first Blender model are represented. These are $3 \times 3°$ tiles, ranging from latitudes of 42° to 48° and longitudes of 336° to 345°. This is confirmed in Figure 7.17a: the tiles are placed about mid-way in the northern hemisphere

[1] available at /http://l2db.selene.darts.isas.jaxa.jp/index.html, last access: 18/04/2018



(a) Full Moon view (b) Close-up

Figure 7.17: Reconstructed CE-3 trajectory

(latitude around 45°) and close to the X_{MCMF}-axis on the negative side (longitude slightly below 360°).

The trajectory reconstruction of the Chang'e-3 landing made by Liu et al. [2014] has horizontal and vertical accuracies of 168.81 m and 58.10 m, respectively, at 9.8 km height. This range should be taken into account when performing the validation.

7.3.2. Image Representability

In the context of this research, a rendered image will be considered representable of the real image, if at least 80% of the N best matches (an algorithm parameter set to 100 for the test) are true matches. This value was chosen taking into account the use of the RANSAC scheme as part of the pose determination (see Chapter 4.4). As discussed in Section 4.4.2, RANSAC does not find the optimal solution. Rather, it finds one that fits a sufficient number of points assumed as inliers. As such, high rates of outliers (in this context, incorrect matches) will result in wrong solutions.

To account for the horizon line and the lander structures, the mask shown in Figure 7.18a was used to limit the area in which features were to be found. In Figure 7.18b, it is shown overlaid on the landing image 10 of CE-3.

The matches were visually evaluated and counted for the real landing images 10, 80, 170, 250, 340, 430, 500, 600 and 700. A match was considered correct when the same structure (*e.g.*, a given crater) was found in the real and data images and the two points are located around the same area in both cases. An example of this last distinction is given in Figure 7.19. The red dots show an incorrect match: despite being associated with the same crater, this hypothetical match would be between different and very distant points of that crater. The characteristics of the data require this subjective judgement.

The first set of rendered images was obtained using the six Kaguya DEMs covering latitudes from 42° to 48° and longitudes from 336° to 345°. The DEMs were patched into one image and applied to a sphere section – corresponding to the aforementioned coordinate ranges – as a displacement modifier in Blender. Figure 7.20a shows the result obtained for the second position of the reconstructed CE-3 trajectory (see Subsection **??**). In this image set, virtually no correct match was found.

The result revealed a very large level of noise in the DEMs. In an attempt to mitigate this, a new DEM was generated by fusing the Kaguya DEMs with an LRO DEM. This last file was generated with Narrow Angle Camera (NAC) images and has a higher resolution and precision. Figure 7.20b shows the second rendered image using the DEM enhanced with LRO data in Blender, with the contrast increased by 75% and bright-

(a) Mask image

(b) Mask overlaid on landing image 10 of CE-3

Figure 7.18: Mask used to remove the horizon and lander structures from the area of detectable features

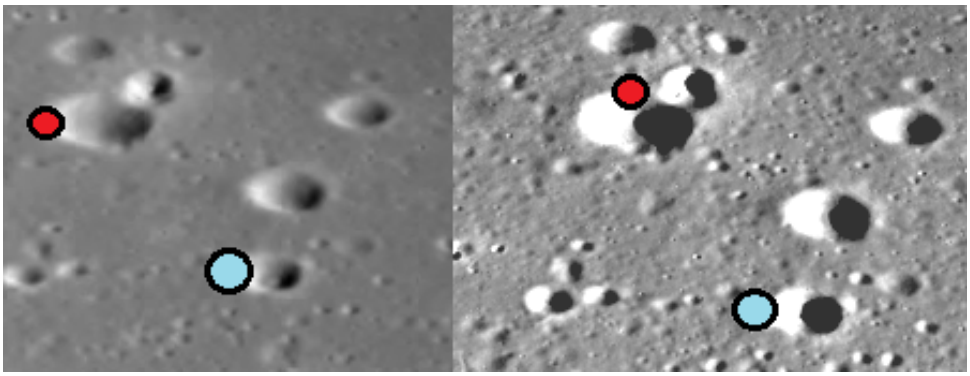
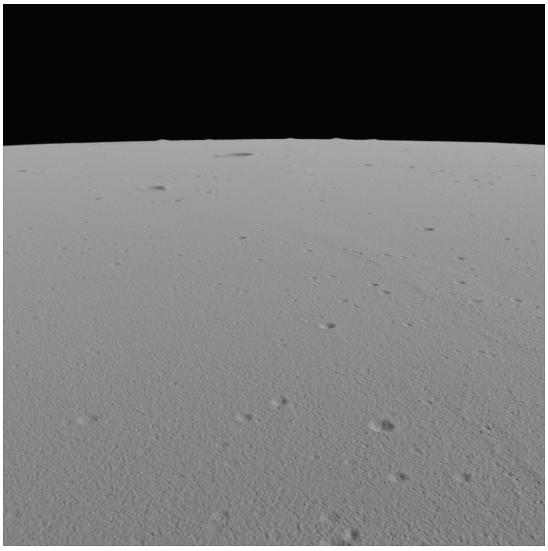
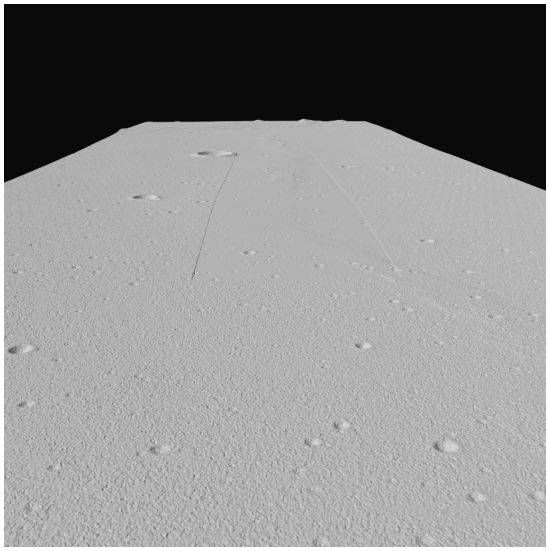


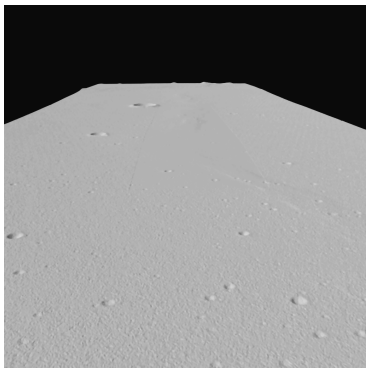
Figure 7.19: Hypothetical correct (blue) and incorrect (red) matches



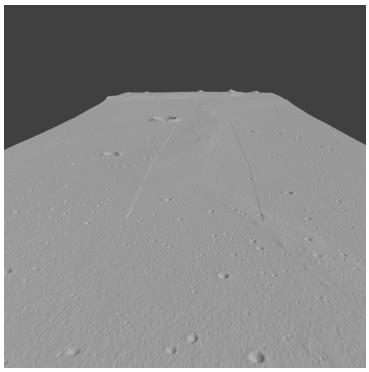
(a) Rendered image using the Kaguya DEMs in Blender



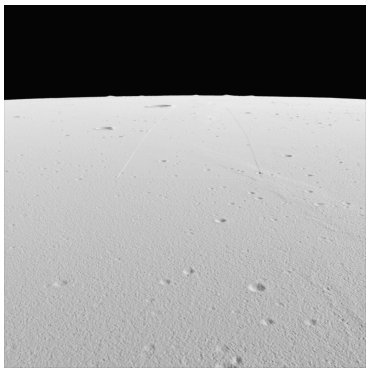
(b) Rendered image using the DEM fusing Kaguya and LRO data in Blender, after luminosity corrections



(c) Using Kaguya and LRO data in Blender, with median filter of size 5



(d) Using the Blender model with smoothing modifier



(e) Using SensorDTM with the fused DEM

Figure 7.20: Rendered images with Blender and SensorDTM for second CE-3 reconstructed position

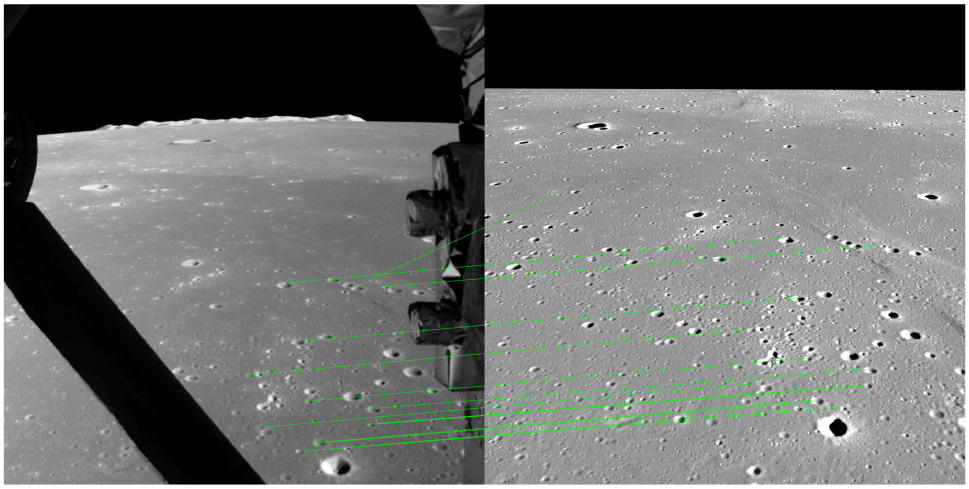
ness decreased by the same amount in *irfanView* (software used for all corrections and filters, unless stated otherwise).

The difference in quality between the two sources of DEMs is clearly visible in this figure, in which the portion corresponding to the LRO data is recognisable and shows much less noise than the remaining area. Still, there wasn't a significant gain in image representability.

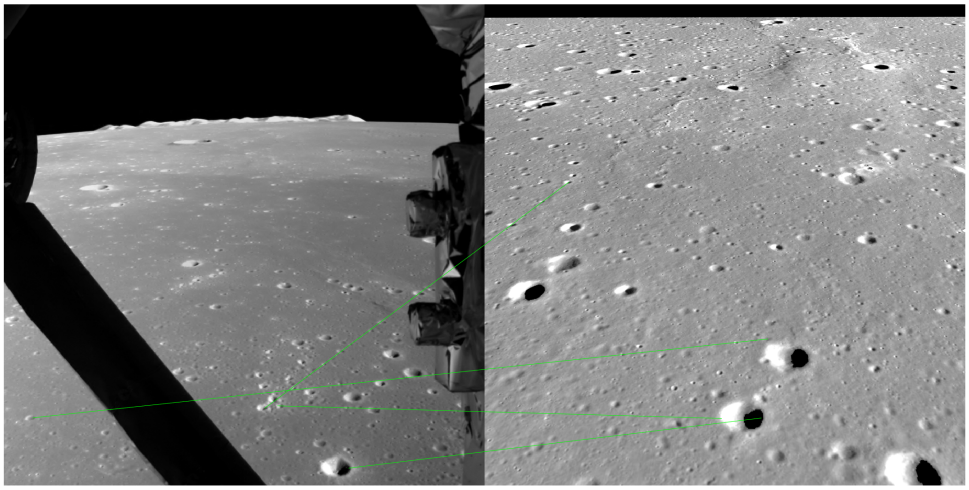
To further reduce the noise, first the last image set was processed with a median filter. Figure 7.20c shows the result using a median filter of size 5. For the second attempt, the result of which is shown in Figure 7.20d, a smoothing modifier was directly applied to the Blender model. None of these sets resulted in significant gains in representability.

Worthy of note are the slight shifts observed between images. It was observed that the positioning of the model in Blender was not very accurate, especially when setting the displacement strength (how much physical displacement should correspond to one bit in the DEM). To obtain a reasonable model, the displacement value had to be around one order of magnitude smaller than the calculated one. For a better model placement, the DLR-internal tool SensorDTM was used (Figure 7.20e). As expected, however, the representability did not improve.

In a final attempt to derive representable images, a new Blender model was created using the Kaguya morning map image as a texture. This was applied to the same spherical section (with the radius changed to correspond to the average height of the area considered). The rates of correct matches ranged from 11% to 34%. Figures 7.21a and 7.21b show matches between the landing image 340 and two different database images ren-



(a) Matches between landing image 340 and texture-based rendered image 7



(b) Matches between landing image 340 and texture-based rendered image 31

Figure 7.21: Examples of matches between CE-3 landing image and texture-based rendered images

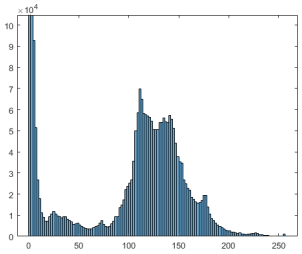
dered with this new texture-based model. The first one shows several correct matches, whereas the matches in the second figure are incorrect.

From the different attempts to improve the representability of the images, the one that resulted in bigger improvements was altering the histogram of the images. This was done by manipulating the values of each bit in the image so that the overall value histogram was more similar to the one seen in the real CE-3 landing images. Figure 7.22 shows an example of these histograms (the scale has been cut for better visualisation). Note that the lower values in the histogram of the real landing image correspond to the lander structures and the space above the horizon. Since these parts are not considered by the feature detector, this part of the histogram is not important.

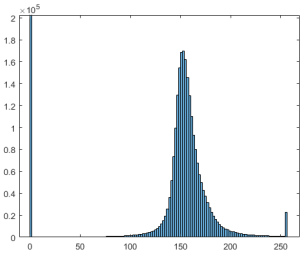
To reduce the contrast of the crater shadows, the minimum value was set to 50. The remaining values, were scalled to fit the range of the real image histogram.

The best rate achieved with this image set was 73% (for landing image 340). The lowest rate was 20%.

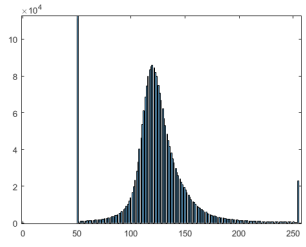
Other image corrections were attempted, namely using median and gaussian filters and adding intermediate points in the trajectory to increase the number of images in the database. None of these efforts yielded representable images.



(a) Real landing image number 340



(b) Texture-based rendered image 7 before histogram correction



(c) Texture-based rendered image 7 after histogram correction

Figure 7.22: Histograms of pixel values

8 | Conclusions and Recommendations

This chapter provides the conclusions obtained from this thesis, both for the object under study, from the results discussed in Chapter 7, and for the research process itself.

In Section 8.1, the conclusions for the presented thesis are given. For future referrence, some lessons learned from the thesis here reported are given in Section 8.2. In Section 8.3, possible directions to expand this research are proposed.

8.1. Conclusions

The research question posed in the beginning of this research was:

What navigation precision can be achieved with an autonomous optical navigation system for lunar landings without an initial state estimate?

The interest in this question arrises from the ever growing need for precision landing for planetary exploration missions. Not relying on initial state estimates allows for use in the *lost in space* context – when no sufficiently accurate estimate of the vehicle's state exists.

This question was broken into three sub-questions:

1. *How accurate is the navigation solution when using ideal data and real images?*
2. *How sensitive is the navigation solution to deviations between the offline data used to generate the database and the real online conditions?*
3. *How representative are the offline images generated through currently available software and DEMs?*

To answer sub-questions 1. and 2., laboratory data was obtained in the TRON lab found in the Bremen DLR facility. This lab was built with the purpose of validating optical navigation algorithms and hardware; and includes three lunar-like terrain models, a KUKA robot to position the camera with high repeatibility and a laser tracker for high precision measurements (more information can be found in Chapter 5). Three different datasets (described in Section 6.2 and Subsection 7.2.1) were obtained:

- **dataset 0** – used to study and tune the algorithm parameters that affect the navigation precision and computation time (see Chapter 7.1); and to validate the algorithm
- **dataset I** – used to answer sub-questions 1. and 2.
- **dataset II** – used to determine the effect of the image area available for feature detection on the navigation precision

The results from dataset 0 complied with the requirements specified in Section 2.5, namely an average error below 1% and a maximum error below 3% (apart from clear outliers), with more accurate results for AKAZE.

In terms of the time per pose requirement, the upper limit was set to 10 s on DLR's optical navigation demonstrator, which roughly translates to 1 s on the desktop used to work on and test the algorithm. The average time per pose obtained was 0.7 s for AKAZE and 1.1 s for BRISK. These measures, however, are very imprecise, since it is only possible to measure the elapsed time instead of the computation time. A precision of about 0.2 s was observed for this particular measurement.

Additionally, as was observed in Section 7.1.1, the methods used within the algorithm can have a significant impact on the computation time. Changing one such method – match downselection – resulted in a reduction of the associated computation time by two orders of magnitude.

Dataset I resulted in much larger errors up to near 20% of the LOS distance with AKAZE (disregarding points with errors above 100%, which were considered outliers). However, upon closer inspection, it was observed that the error was mostly given as a function of the image area available for feature detection. This hypothesis was supported by the results obtained with dataset II, which consisted of the same initial camera positions as dataset I, but using a camera lens with a smaller FOV, which increased the search area for the same camera pose.

This dependency between the position error and the search area is explained by the pose determination method used. EPnP begins by determining the camera orientation and using the result to calculate the camera position – resulting in the correlation observed in Subsection 7.2.4. If the points used to calculate the orientation are very close to one another (in the case of a small continuous area), a small deviation in one of these

points will result in very large rotation errors. When the points are further apart (using a bigger continous area or various small areas dispersed in the image), the consequence of such deviations is smaller. This phenomena was also used in Section 3.5 to justify the pre-selection of features based on their position in the image.

In reality, as observed with the CE-3 landing images, the search area can be limited by the presence of lander's structures and by the horizon line. On one hand, this characteristic can be taken into account during mission design to position the camera such that the area occupied by structures of the lander in the image is minimised. On the other, the horizon line will only reduce the area range in one axis (*e.g.*, the Y_C-axis) while maintaining the full range of the other one. In such a case, it would be likely to observe a bigger error in one direction (associated with the reduced range axis) than the other.

During vertical descent, apart from the lander's structures, the search area would always be 100%, for which, in both AKAZE and BRISK, the position error was within the required range. Additionally, the results from dataset I with over 50% search area show a smaller error for AKAZE than for BRISK, which is in accordance with the results derived from dataset 0. Due to the limited number of available points in such conditions, however, it is not possible to generalise these conclusions using dataset I alone.

As a consequence of this area dependence, the errors observed in dataset I were mostly caused by the small search area available. Thus, as a whole, it is not a reliable source of data to determine the effect of trajectory disturbances. Dataset II was obtained, in part, to serve this purpose. However, it is possible that the search areas were still too small to sufficiently reduce their effect (below 50%).

Alternatively, using only the points of dataset I for which the search area is higher than 60%, no relation was observed between the error and the distances from the disturbed point to the paired nominal one. The pair was made by choosing the nominal point situated in the closest $X_M Y_M$-plane – smallest difference in z_M-coordinates. The distances used as measures of the disturbance were the z-distance – difference in z_M-coordinates – and the projected distance – disregarding the z_M-dimension – both normalised with the LOS distance. This independence can be explained by the invariances associated to the feature detectors, namely translation, rotation and zoom invariance.

It should be noted that the algorithm is highly non-linear and, as a consequence, the effects of a change in the camera position in the navigation error cannot be determined *a priori*. For example, in a different position, the features detected and matched may or may not be different; and the RANSAC scheme may or may not pick the same inliers. These effects have a strong impact and are both not strongly dependent on translation or rotation of the image. On the other hand, they are strongly affected by the algorithm parameters used, as observed during the parameter tuning in Chapter 7.1.

The proposed hypothesis regarding the sensitivity of the algorithm is that it will be mainly affected as a function of the common area between the input image and the database images. On one hand, if only correct matches are used, the orientation error effect described above will reduce the navigation precision for small percentages of common area. On the other hand, since there will still be terrain on the surrounding area, wrong matches are possible, especially when using the image division described in Section 3.5. However, for the expected disturbances, a high percentage of common area is expected. Otherwise, it may still be possible to enhance the database with additional images.

In sum, the algorithm was mostly sensitive to the available search area. A search area above 60% leads to navigation solutions mostly within the requirements specified. For AKAZE, the outliers are likely to be detectable by filters. Sensitivity to the distance from a disturbed trajectory point to the closest nominal trajectory point is either inexistent or was hidden by the larger effect of the search area. Disturbances in the trajectory *per se* do not have a strong impact on the navigation error, thanks to the feature detectors' invariances to image translation, rotation and zoom.

To answer sub-question 3., the landing images from CE-3 and the corresponding estimated trajectory (described in Subsection **??**) were used. Images were rendered using two rendering softwares – Blender and SensorDTM (DLR internal tool) – and two different DEMs – from the Kaguya and LRO missions. Various methods were attempted in Blender.

None of the data image sets generated with the DEMs were found to be representable of the real images (over 80% correct matches chosen between the data images and a real image, as defined in Section 7.3). The DEMs (the Kaguya ones in particular) were found to be too noisy for effective use.

A final attempt was made using the Kaguya morning maps as texture images in Blender. Various attempts in processing the rendered images were made, namely histogram adjustments and median filters. These sets resulted in a much higher correct matching rate. Nevertheless, the rate was highly variable between images of a given set and did not consistently reach the required threshold.

8.2. Lessons Learned

Having conducted this study in a company environment with both software and laboratory work, this thesis was very conducive to a wide variety of experiences and learning opportunities. The most notable difficulties, surprises and successes were used to derive the following list of lessons learned.

- **Software**
  - Name everything very clearly (files, folder, ...)
  - Either keep only one version of a script or be aware of the different existing versions
  - (Try to) stay organised and document as you go
- **Data processing**
  - Keep notes on the hardware used to acquire a given set of data; the date of acquisition; and other relevant information
  - Always check any configuration file before running a tool
  - Run long simulations and searches on local drives
  - Build long simulations and searches such that the intermediate results are saved during the run and it doesn't have to be rerun from scratch if stopped
  - When running a long process, check any intermediate output as early as possible
- **Laboratory work**
  - Take into account as many limitations of the lab as possible
  - Be prepared for the unexpected and the unexplained
  - Reboot camera, laser tracker and KUKA commanding software
- **Working with others**
  - When depending on others' input, be clear about your schedule and how it fits with theirs
  - Acquire data as early as possible, to avoid having schedule incompatibilities delaying your work
  - Be clear about what you know and what you don't
  - Keep an open mind on others' perspectives, while defending your own opinion
  - Keep track of meeting outputs, commitments and next steps
- **Thesis specific**
  - Always compare a rendered image with the corresponding undistorted one for each set
  - Don't wait for the rendering process to finish to check the outputs
  - It may be beneficial to include extra lamps for specific lighting conditions
  - Invest time to properly set the camera exposure and resolution before starting data acquisition (especially for camera calibration images)
  - Plot the frames involved after every frame transformation to check for errors

8.3. Recommendations for Future Work

Following are some suggestions for future work to expand on this thesis, which, either due to time constraints or for being outside the limits imposed, have not been addressed.

1. Check similarities between descriptors within a database
2. Explore different feature detectors and new methods for feature and match downselection
3. Perform sensitivity analysis for realistic search areas and relate errors to percentage of common area
   - If necessary, explore database enhancement to account for expected disturbances
4. Test the algorithm in the optical navigation demonstrator
5. Optimise the algorithm and characterise the computation time and processing load
6. Integrate the algorithm within a GNC system and test the landing accuracy
7. Test the algorithm for illumination robustness in the context of an asteroid landing
8. Explore new rendering methods with the real DEMs and image maps

The first recommendation is to study how similar the features in the database are to one another and what impact that has on the navigation accuracy. It is expected that, if the features in the database are too similar to one another, it is more likely for incorrect matches to occur during the online script. Thus, selecting the database features based on their similarity may increase the navigation accuracy.

Recommendation 2. is based on the observation that the performance of the algorithm was affected by the detector used. Namely, BRISK yielded less consistent results in comparison to AKAZE. Therefore, it is possible for other detectors to perform better for a lunar landing scenario. Furthermore, new methods can be designed for the feature and match selection – for example, using the idea described for the first recommendation. To select the features, a minimum threshold could be set on the distance between the descriptor of a feature and

its NN in the rest of the detected features. To select the matches, an online point could be matched to its first and second NNs. Then, the ratio between the distance to the first and to the second could be used as a measure of how likely the match is correct.

The third recommendation is based on the proposed hypothesis that the navigation algorithm will be sensitive to disturbances in the trajectory insofar as this reduces the common area between the images captured at the nominal and disturbed points. In other words, a relation between the navigation accuracy and the common area between the database images and the online images is expected. The sensitivity analysis should include only images with nearly 100% of the image area available for feature detection.

Once this relation has been found, in case the algorithm is found to not be robust to the deviations expected during a landing mission, it is recommended to explore the addition of extra images for the generation of the database. This enhancement should be studied in terms of the improvement in accuracy and robustness, and the effect on time and memory load.

To validate the use of the algorithm in real applications, it should be tested in DLR's navigation demonstrator. This should determine whether the algorithm can be used in real-time.

Recommendation 5. stems from the recognition that the script has not been written with time performance in mind. The optimisation of the algorithm for faster computation times should be performed. Furthermore, the time performance and memory load should be characterised with higher precision.

Noting that the ultimate goal of the algorithm is to allow PPL, the achievable landing accuracy should be tested. To do so, it should be integrated into a complete GNC system to be simulated for a lunar landing mission.

To check the applicability of the algorithm for asteroid landings, it is recommended to check its robustness to variability in the illumination conditions, given the added uncertainty, not present for lunar landings.

Finally, methods to obtain appropriate images using real planetary data (such as the Kaguya and LRO DEMs and lunar image maps) should be explored, to check the applicability of the algorithm in real landing missions. Additionally, a more precise measure of representability should be defined. This measure should be determined with respect to the final navigation accuracy.

A | Work-flow

This appendix provides detailed instructions for the use of the developed tools in the appropriate order. Section A.1 provides instruction to build the Blender models for the TRON terrains. Although a working template with functioning nodes (for depth rendering) is made available, Subsection A.1.1 details the steps used to create them.

The next step is the calibration of the camera, which is explained in Section A.2, namely the aquisition of the images in the TRON lab and the use of the DLR developed calibration tools, CalDe (Subsection A.2.1) and CalLab (Subsection A.2.2).

Section A.3 describes the steps in the TRON lab to aquire data for a given trajectory. Sections A.4, A.5 and A.6 instruct on how to use these data and the camera calibration results to generate the trajectory file, the image directories and the camera parameters file, respectively.

Section A.7 explains the use of the A1 script (Subsection 6.3.2) to render the images and depth data necessary to generate a database. The changes in Blender required for the depth data rendering via the command line are described in Subsection A.7.1. Section A.8 provides instructions to generate the masks to limit the feature detection area in an image.

Sections A.9 and A.10 provide the inputs and commands required to run the offline and online scripts, respectively. Finally, Section A.11 explains how to obtain the accuracy performance plots from the original trajectory and the navigation solution.

A.1. Building TRON Blender Models

To create a Blender model for the TRON terrains, the associated DEM and auxiliary text file should be used. These data should be readily available and are a result from scanning the intended terrain model with the laser scanner available in the TRON lab.

The first step is to generate a text file used to set the dimensions and place the model in Blender. It also provides the dynamic range necessary to set the Blender displacement modifier's *strength* (how much one bit of the DEM corresponds to). To do so, the script *write_Blender_info_file.m* should be run, after changing the values of the *DEM_image*, *DEM_aux_file* and *output_info_file* in the code section *Input variables* accordingly. The paths given can be both relative or absolute paths.

Next, a new Blender file should be created by copying the file *model_template.blend* (to render only images) or the file *model_template_depth.blend* (to allow the rendering of depth data). Figure A.1 shows its layout. This file contains a script (marked by the green box in Figure A.1) that automatically creates and places the model according to the provided DEM image and info file. The variable *max_subdivisions* controls the resolution of the 3D model and should only be decreased in case of computational limitations or if the resolution of the DEM is much higher than require to render the images. The paths to these files should be given as values for variables *terrain_model_dem* and *terrain_model_info*. Begin relative paths with '\\'. In case the script is not visible, the screen layout should be changed to *Scripting* in the appropriate menu at the top of the Blender window.

Once the values of these variables have been introduced, check if the rendering engine is set to *Cycles Render* and run the script by clicking the appropriate button, at the bottom of the script window. In case of an error, the error message can be seen by toggling on the system console: select the *Window* menu, followed by *Toggle System Console*.

Note that for an unknown reason, the DEMs generated from the laser scanner data cannot be used directly in Blender. Doing so, causes the programm to crash. To correct this, the original *tiff* file was read in MATLAB and rewritten with the same format. The new files can be used succesfully.

Once the script as been successfully run, the model should be visible in the right window (*3D View*). If necessary, to centre the view on the model, click the '.' key on the numberpad, with the cursor hovering on this window [1]. Apart from the model, a camera named *camera* and a lamp named *Point* should have also been added.

[1] In Blender, pressing a key will allways affect the window in which the cursor is hovering

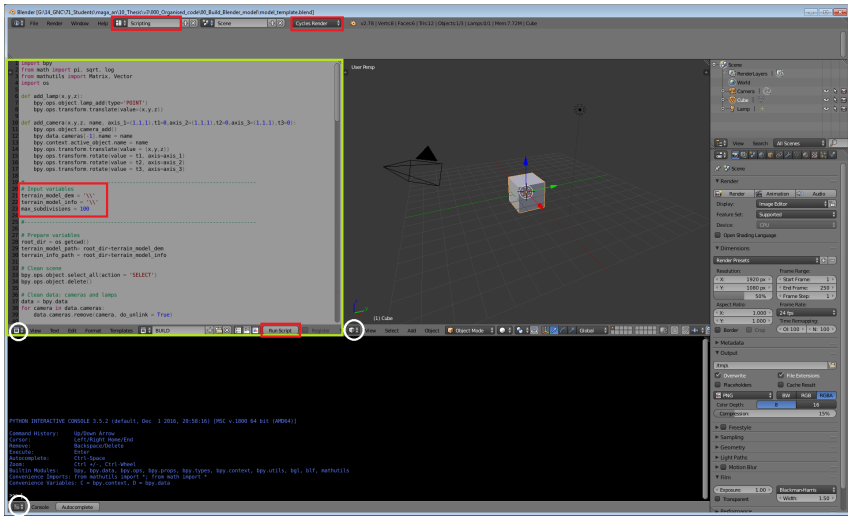


Figure A.1: Blender file layout

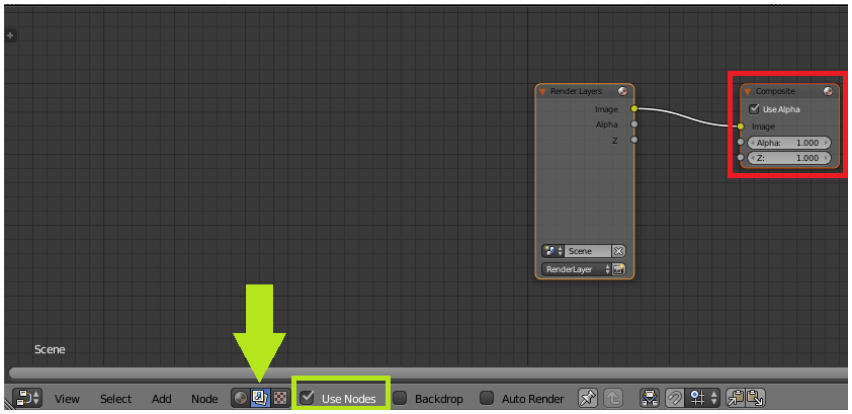


Figure A.2: Initial compositing nodes layout

A.1.1. BLENDER NODES

In this section, the process used to create the nodes required for depth data rendering is described. This has already been done for the file *model_template_depth.blend*.

Open a node window by clicking on the bottom left symbol in any of the current ones (white circles in Figure A.1) and chose the option *Node Editor*. The current window will change into a node one. In the bottom of this window, click on the *Compositing nodes* option (green arrow in Figure A.2 and select the option *Use Nodes* (green box). Next, delete the *Composite* node (red box) by selecting it with the left mouse button and pressing the *delete* key.

Add a new *File Output node* by clicking Shift-A (with the cursor hovering the nodes window), selecting *Output* and *File Output*. Click the left mouse button to place the node in the window. With this node selected, change the properties of the node in the right side of the window, to allow the rendering of the depth data (Figure A.3). In case this menu is not visible, press the 'N' key.

The process is described below. Each step is shown in Figure A.3.

1. Change the *File Subpath* to *image_*
2. Click *Add Input*
3. Select the new input in the box below the button and change the *File Subpath* to *zpass_*
4. Choose the *PNG* file format for the node
5. With the *zpass_* output selected, unselect the option *Use Node Format*
6. Select the *OpenEXR* file format for the output
7. Choose the *Float (Full) Color Depth*
8. Choose option *None* for *Codec*

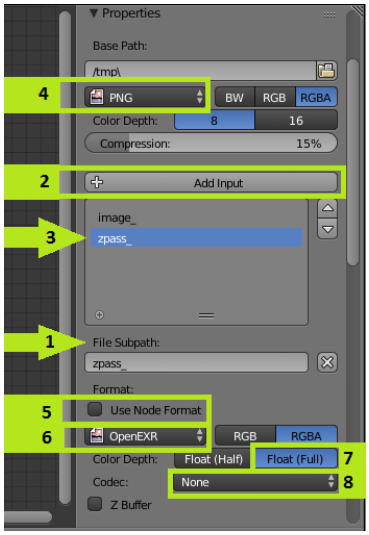


Figure A.3: Output node properties

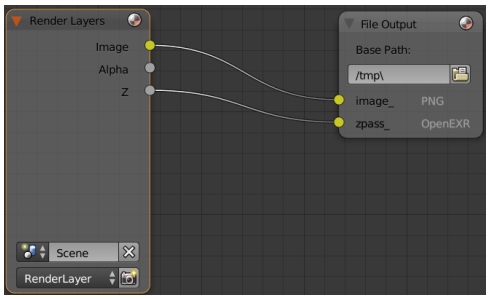


Figure A.4: Final node layout for image and depth data rendering

Finally, click on the yellow circle of the *Render Layers* node (*Image*) with the left mouse button and drag to the *image_* input of the *File Output* node. Do the same with the *Z* output of the *Render Layers* node to the *zpass_* input of the *File Output* node. The result is shown in Figure A.4.

A.2. Camera Calibration

Every time the camera and/or the T-MAC is moved with respect to one another or to the KUKA Tool Centre Point (TCP), a new camera calibration has to be performed. Thus, the first step is to place the camera and T-MAC in the KUKA arm with the position intended to perform the trajectory measurements.

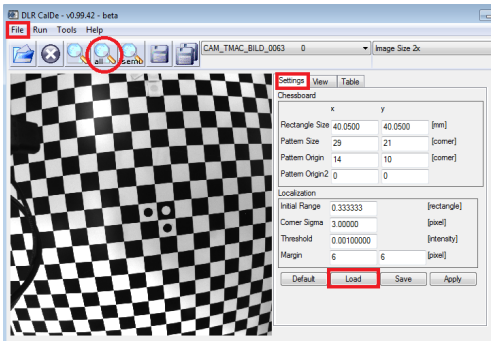
Next, the calibration target is placed in the lab floor in a location that allows measurements of the T-MAC by the LT. A script with a pre-defined set of TCP positions can be run, which places the camera to capture the calibration images. During this run, the T-MAC measurements for each position are taken and used to derive the transformation matrix from the LT to the T-MAC. This matrix is saved in a *frame* file with the same name as the corresponding image.

Alternatively, this process can be done manually, by placing the robot in the desired position, capturing the image, performing the T-MAC measurement and, later, using these measurements to obtain the required matrices.

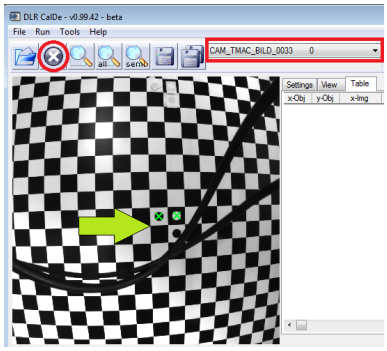
With these outputs, the calibration can be performed using CalDe (Subsection A.2.1) and CalLab (Subsection A.2.2). Before performing the calibration, images that are too similar should be removed to avoid skewing the results. Both the tools mentioned are run through an Interactive Data Language (IDL) virtual machine.

A.2.1. Camera Calibration Detection Tool (CalDe)

Run the *calde.sav* file. Load the calibration images by clicking *File*, followed by *Load image(s)* (use the *Shift* or *Ctrl* key to select multiple images). The images should be in the same folder as the corresponding *frame* files

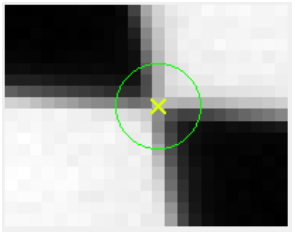


(a) Loading images, configuration file and searching points in all images

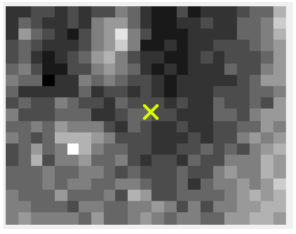


(b) Semi-automatic option for finding points

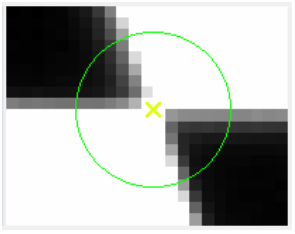
Figure A.5: CalDe initial set-up



(a) Good detection



(b) Misplaced detection



(c) Over-exposed corner

Figure A.6: Example of point detections

to include that information in the output files (required for extrinsic calibration). In the *Settings* tab, click *Load* and chose the appropriate *clg* configuration file for the calibration target used. Click on the *all* button (red circle in Figure A.5a) to detect the points in all images. This process can take several minutes.

In case the points are not sucessfully detected automatically for a given image, select that image from the drop down menu with the image names (red box in Figure A.5b) and click the *semi* button. Then, click on the three dots of the calibration target representing the origin (the order is not important). If the points are not detected, remove the image by clicking on the 'X' button (red circle in Figure A.5b).

Once the points have been detected, select the *Table* tab. Click on a table cell to see a close-up of the detected point in the small window under the table. Find bad detections and either click in the original image near the correct location of the point (a new location will be determined automatically) or delete the point by right-clicking on the cell and choosing *delete*. Figure A.6 shows examples of good (Figure A.6a) and bad detections.

Figure A.6b is an example of a misplaced point. This kind of bad detection may be corrected by relocating the detection (as previously described). On the other hand, Figure A.6c is an example of over-exposure, where the corner is not clear. Points such as this should be deleted. The green circles around the detected point in both the original and close-up images represent the uncertainty associated to the detection (the bigger the circle, the higher the uncertainty).

Finally, save the image points by clicking the cassete symbol buttons (the single cassete to save the current image's points or the button with two to save the points of all images), or through the *File* menu. The output *pts* files should be saved in the same folder as the images and *frame* files.

A.2.2. Camera Calibration Lab (CalLab)

Once again, start by running the *callab.sav* file and load the same images used to detect the points in CalDe (include only the images for which points were detected). Next, load the points through the *File* menu. In the *Settings* menu, click on *Intrinsic camera parameters*, and choose the order of distortion to consider (Figure A.7).

Next, under the *Run options* menu select *Run stage [1]*. This will perform the intrinsic camera calibration. Once the iteration is finished, histograms of the pixel errors for the reprojected points in comparison to the

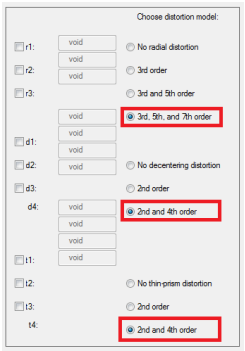
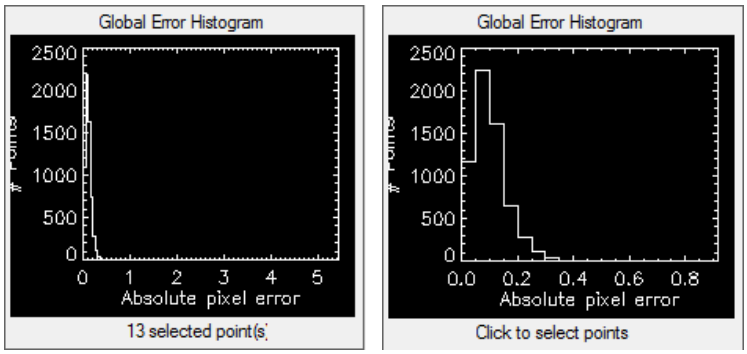


Figure A.7: Distortion model to use for camera calibration in CalLab



(a) First iteration (b) Second iteration

Figure A.8: Histograms of CalLab intrinsic calibration

detected ones (CalDe output) are provided. To refine the calibration, click on the *Global Error Histogram* in a position corresponding to the desired pixel error. This will select all the points for which the error was higher than that error (values to the right of the point clicked in the histogram). Then, delete the points by pressing the *delete* key and rerun the same option. Be aware not to delete too many points, since this can lead to inaccurate results (even if the pixel errors are smaller). Figure A.8 shows the resulting histograms from two consecutive iterations.

Once the desired error has been achieved, select *Run stage [2]* under the *Run options* menu to perform the extrinsic calibration. The same refinement can be made for this case, although not recommended (unless the intrinsic calibration is recomputed). Having finished the calibration, save the output through the *File* menu by selecting the option *OUTPUT: Save DLR-RMC calibration file*.

A.3. Acquire Lab Data

Once the camera has been calibrated, the same camera-T-MAC set-up can be used to capture the descent images and the corresponding T-MAC measurements. To do so, a *xml* file containing the TRON commands for the TCP positions and orientation is used. The robot moves according to the commands, an image is captured and the corresponding T-MAC measurement is taken.

Apart from these measurements, the LT should also be used to measure the terrain model's reflectors and the approximate position of the lamp. To measure the lamp position, the reflector needs to be held behind it, at the point that could be thought as the origin of the light rays. The first are 3D static measurements, whereas the second can also be a 3D continuous measurement, which is later averaged, in case the person holding the reflector cannot do so steadily enough. These measurements should be made before the images are taken, especially for the lamp, since it can be accidentally moved during the position measurement.

```
0   Traj_Id_01051_2018-03-28    Traj_Id_01101_2018-03-28    Traj_Id_01001_2018-03-27
1   -   -   Traj_Id_01002_2018-03-27
2   Traj_Id_01053_2018-03-29    Traj_Id_01103_2018-03-28    Traj_Id_01003_2018-03-27
4   Traj_Id_01055_2018-03-29    Traj_Id_01105_2018-03-28    Traj_Id_01005_2018-03-27
5   -   -   Traj_Id_01006_2018-03-27
7   Traj_Id_01058_2018-03-29    Traj_Id_01108_2018-03-28    Traj_Id_01008_2018-03-27
```

Figure A.9: Example of file containing the trajectory sections information

Table A.1: Format of camera pose data in trajectory file

pose id	Camera position			Camera quaternion				Lamp/Sun position			Lamp strength/exposure
	x_C^M	y_C^M	z_C^M	q_0	q_1	q_2	q_3	x_S^M	y_S^M	z_S^M	ϵ

A.4. Generate Trajectory File

With the reflectors', lamp's and T-MAC measurements files and the camera calibration file, the trajectory file can be generated via the script *generate_trajectory_file_TRON.m*.

To account for multiple trajectory sections, a text file should be used specifying the sequence to use. Each line corresponds to one trajectory. The first column should have the id number of the trajectory. In each of the remaining columns, the name of the folder containing the TRON data for that section should be included. If a given trajectory has no data for a given section, the symbol "-" should be used. The columns should be separated by tabs. There is no limit to the number of sections that can be used. An example of such a file is shown in Figure A.9.

Apart from this file and the camera calibration file, the data should be organised as follows:

- **Lamp data** – folder containing:
  - *LT_lamp.dat* – file containing the LT measurement of the lamp position
  - *LT_reflectors_lamp.dat* –file containing the LT measurement of the terrain model reflectors with the LT placed at the same position as used to measure the lamp
- **Trajectory data** – folder as outputed by the script used in TRON to run the trajectory; should include the folders *images* and *logs*; and:
  - *LT_reflectors.dat* – file containing the LT measurement of the terrain model reflectors with the LT placed at the same position as used for the trajectory run (to be added by the user)

In the code section *Input variables*, change the variables to reflect the paths to the data files. The *lamp_strength* variable is set arbitrarily and needs to be tested [2]. The output is a *txt* file that can be used in Blender, to generate the database and to analyse the accuracy of the navigation solution. The file structure is shown in Table A.1, where (x_C^M, y_C^M, z_C^M) and (x_S^M, y_S^M, z_S^M) are the camera and Sun position, respectively; and ϵ is the lamp strenght (in Blender) or exposure time (in SensorDTM). This file format can also be used with SensorDTM (the use of which will not be described here).

A.5. Generate Image Directories

The script *generate_image_directories.m* is used to facilitate the organisation and naming of images. The same text file containing the trajectory sections and respective folders should be used. The original images will be copied into the desired output folder with the name format required by the navigation algorithm.

A.6. Generate Camera Parameters File

Two camera parameters files must be generated using the *generate_camera_parameters_file.m* script: one corresponding to the real camera; the other to the camera simulated in Blender. For the first one, the variables are taken from the camera calibration file. The values for the variables are summarised in Table A.2. The non-mentioned variables can be set to *nan*. Note that only square images are currently considered (the width and height of the images should be the same).

For the simulated camera parameters file, the same resolution should be use. The *dist* variable is set to

[2] Use one of the camera poses from the trajectory and render the same image multiple times with different lamp strength values until the lighting obtained is satisfactory

Table A.2: Correspondence between variables in the *generate_camera_parameters_file.m* script and the camera calibration file

Script	Camera calibration file
vars('resolution')	camera.0.width
focal_length_mode	0
K	camera.0.A
dist	[camera.0.k1, camera.0.k2, camera.0.p1, camera.0.p2, camera.0.k3]

nan. The **K** matrix is the closest ideal matrix to the real one, meaning

$$\mathbf{K} = \begin{bmatrix} f & 0 & \frac{resolution}{2} \\ 0 & f & \frac{resolution}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

where f is the closest integer to the first two diagonal entries of the real **K** matrix. Although this number need not be an integer, it is adviced to keep the precision low, so as to avoid inconsistencies with the actual camera used by Blender (since it is not clear what the maximum precision allowed is).

The variable *vars('K_mode')* should be set to 1 and *focal_length_mode* to 0. Variables *vars('focal_length_mm')* and *vars('pixel_size')* should be such that dividing the first by the second is equal to f (used in Equation A.1).

A.7. Render Images and Depth Data

To speed up the rendering process, the image rendering was performed using a virtual machine. Create a folder with the A1 script; the terrain model and DEM used to build it; the simulation camera parameters file and the trajectory file. Copy this folder into the virtual machine account using the command *scp -r* in a *cygwin* command line. Access the account using the command *ssh*.

Navigate to the folder containing the A1 script and run the command *blender {relative path to 3D model} -b --python A1_render_images_depth.py {root directory absolute path} {output folder (path from root)} {camera parameters file (path from root)} {trajectory file (path from root)} {render depth data (0/1)}*. In the end, the output folder can be copied back into the regular environment using the *scp -r* command.

To check whether the rendering was performed correctly, undistort the real images captured in TRON and visually compare them with the rendered ones. The real images can be undistorted using the script *undistort_images.py*.

Before running it, set the *input_folder_name* variable to the folder containing the original images; and the *output_folder_name* variable to the desired output folder (will be created automatically if non existent). Set the variable *A_input* to the real **K** matrix (from the calibration) and *A_output* to the **K** matrix in the simulation camera parameters file used to render the images. Finally, set *distortion_coefficients* to be the same as *dist* in the real camera parameters file.

A.7.1. Adapting and Building Blender

To be able to render the depth data through a script, the source code for Blender has to be changed [3]. The changes are:

- **file**: source/blender/compositor/operations/COM_ViewerOperation.h; **line**: 58
  - **from**: *bool isOutputOperation(bool /*rendering*/) const { if (G.background) return false; return isActiveViewerOutput();}*
  - **to**: *bool isOutputOperation(bool /*rendering*/) const {return isActiveViewerOutput(); }*
- **file**: source/blender/compositor/operations/COM_PreviewOperation.h; **line** 48
  - **from**: *bool isOutputOperation(bool /*rendering*/) const { return !G.background; }*
  - **to**: *bool isOutputOperation(bool /*rendering*/) const { return true; }*

Then, Blender can be built using cmake [4].

[3] /https://blender.stackexchange.com/questions/69230/python-render-script-different-outcome-when-run-in-background, last access: 18/04/2018

[4] See /https://wiki.blender.org/index.php/Dev:Doc/Building_Blender for more detailed information, last access: 18/04/2018

```
19 #------------------------------------------------
20 # Input variables
21 terrain_model_dem = '\\model3.tiff'
22 terrain_model_info = '\\info_model3.txt'
23 max_subdivisions = 0
24
25 #------------------------------------------------
```

Figure A.10: Location of the *max_subdivisions* variable in the build script

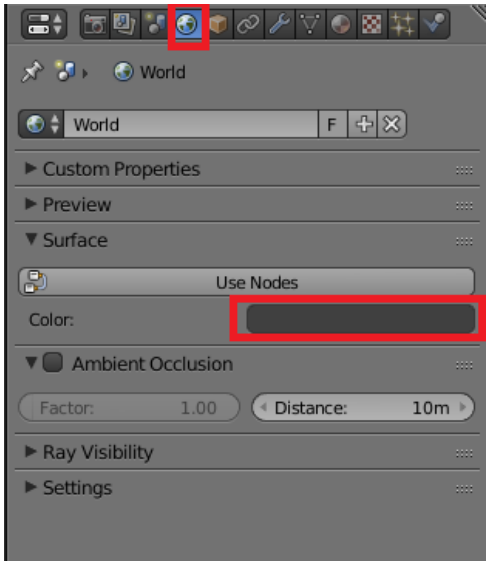


Figure A.11: Location of the *World* icon and the *Color* rectangle

A.8. Generate Feature Masks

To use a constant feature mask for all images (as done for the CE-3 landing images), create the desired mask, name it *mask.png* and store it in the appropriate data folder.

To use Blender to render the mask images, start by creating the appropriate model:

- Make a copy of *model_template.blend* and set the terrain model variables accordingly
- Set the variable *max_subdivisions* in the script to 0 (see Figure A.10)
- Build the model by running the script
- Click on the *World* icon and then on the *Color* rectangle (see Figure A.11)
- Set all RGB values to 0
- Save the model

With this new model, render the masks using the trajectory corresponding to the images to be masked. Change the lamp strength in the trajectory file to a very high number (*e.g.*, 100,000,000) and render the images.

The rendered masks can be used with the data(online) images. To apply the same masks to the online images, they should first be distorted using the command *distort_images.py* -- {real camera parameters file} {simulated camera parameters file} {rendered masks folder} {output folder}.

The folder containing the feature masks should be named *masks* and placed in the folder containing the rendered images or the online images accordingly. Note that it is not necessary to use a mask in every image; in case no mask is needed for a given image it can be removed. An example of such a case is when all the area in the image is suitable for feature detection. This would result in a fully white rendered mask. The distorted mask, however, could have some black portions near the edges.

A.9. Generate Database

To generate the database, choose, from the rendered images, the ones to use. Do not change the original trajectory file, since the order is associated with the indexes used to name the images. Instead, create a new folder (render folder) with the subfolders *rendered_images* and *zpass*. Copy the intended images to the first subfolder and the associated depth data files to the second one. Also include the simulated camera parameters

file.

To run the offline script, either use the *run_A.bat* file or the command *A_generate_database.py -- {render model [-]} {render folder} {render images (0/1) [0]} {camera parameters file} {trajectory file} {output database folder} {index file name} {world points file name} {descriptors file name} {feature detector} {image division} {percentage non div} {n data features} {precision} {threshold data} {octave} {descriptor size (if AKAZE)}*.

If the images have been already rendered (as is the case discussed here), *render images* is set to 0 and *render model* can be set to '-'.

A.10. Get Navigation Solution

The required inputs to run the online script are the database generated; the real images; and the real camera parameters file. The command to use is *B_determine_camera_pose.py -- {images folder} {images name root} {detector} {database folder} {index file name} {world points file name} {descriptors file name} {real camera parameters file} {output file name} {divisions} {percentage non div} {n query features} {n best matches} {threshold query} {descriptor size (if AKAZE)}*.

The output will be of the same format as the trajectory file (Table A.1), except for the lamp variables which are missing. Also, the quaternion refers to the $\mathscr{F}_C$ frame instead of the $\mathscr{F}'_C$ one.

A.11. Analyse Accuracy

To analyse the accuracy of the navigation solution, the output of the online script and the trajectory file associated to the real trajectory are used as inputs. Note that the trajectory file does not have to be the one used to render the database images. In fact, this is not the case when analysing the sensitivity of the algorithm to disturbed trajectories (Subsection 7.2.3). Nevertheless, they can be generated using the same process described in Section A.4 (without concern for the lamp variables).

To perform this analysis, run the script *analyse_performance.m* after appropriately changing the input variables, namely the path to the trajectory file and to the navigation solution one. The script plots seven histograms, namely for the norm of the position error; the position error along each of the axis; and the orientation error between each of the axis. These plots are automatically saved if the variable *save_plots* is set to 1.

B | Optimised Trajectories

The trajectories optimisation made through the software described in Subsection 6.4.3 returned a nominal trajectory and 50 disturbed trajectories. These are meant to place the TCP of the KUKA robot in the TRON lab to capture the necessary images, later used to derive a navigation solution.

The nominal trajectory had an initial position equal to $\begin{pmatrix}-101 & 50 & 1500\end{pmatrix}^T$ and initial velocity equal to $\begin{pmatrix}10 & -10 & -40\end{pmatrix}^T$ in the reference system centred in the landing target, where the Z-axis is perpendicular to the landing surface. The disturbed trajectories were generated with the same software by changing the initial state with random uniform error distributions of 100 m of radius from the initial state and 10 m/s for the velocity magnitude. These initial states are shown in Figure B.1, where the cyan dot represents the nominal initial state, and the axis and dashed lines mark the range within which the disturbed states can be generated (100 m in each direction from the nominal state). From the 50 disturbed trajectories, 20 were selected based on the spatial constraints of the lab and their mutual similarities. The initial states for these trajectories are marked with red dots, whereas the blue crosses represent the initial position of rejected the points.

Before running the trajectories in the TRON lab, they were converted from the above described reference frame into KUKA commands (Section B.1). Next, the constraints associated with the LT were used to find adequate positions for the instrument and to section the trajectories accordingly (Section B.2). Finally, Section B.3 elaborates on the unexpected issues that arose in the lab.

B.1. Frame Transformations

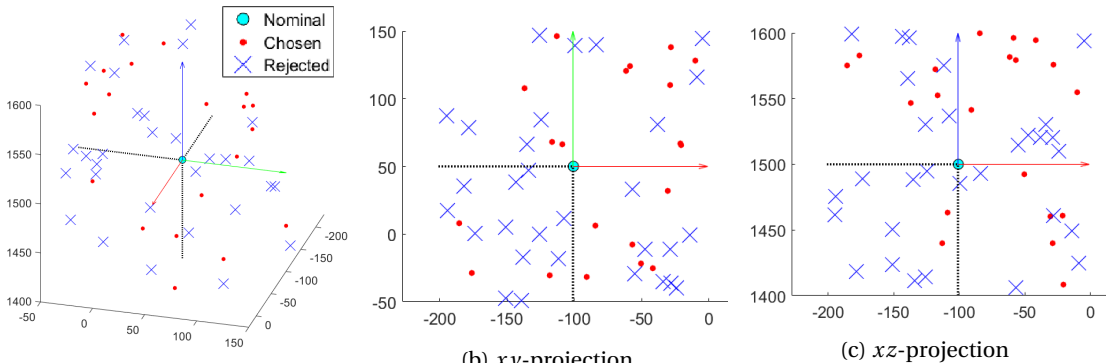
The first step in transforming the output from the trajectory optimiser to KUKA commands is to scale the trajectory.

For a realistic landing scenario, a scale of 1:100 was chosen. With this scale, the smaller "rocks" would have a diameter of about 50 cm in the real scale, thus small enough not to constitute a landing hazard in most missions. In the centre of the terrain model, there is an area of about 20×20 cm (thus 20 m in real scale) where no bigger rocks or craters exist. This was considered to be the landing site, since the area is bigger than the safe radius of 9 m (three times the diameter for a lander of 2 m).

The chosen landing site was located at $\begin{pmatrix}0.8 & 1.9 & 0.2\end{pmatrix}^T$ in the $\mathcal{F}_I$ frame. This point is shown in Figure B.2. This point was moved from the centre towards the left due to the restrictions imposed by the LT (see Section B.2).

After scalling and shifting the trajectory, the result was given in the $\mathcal{F}_M$ frame. The coordinates and orientation were first transformed into the $\mathcal{F}_{TRON}$ frame and then to the $\mathcal{F}_{KUKA}$ frame – used to move the robot – through the appropriate transformation matrices provided.

Apart from these transformations, the orientation of the TCP required a correction of 30° in the roll angle, due to the addition of an adapter at the end of the robot arm. The final position and orientation of the TCP



(a) 3D view

(b) xy-projection

(c) xz-projection

Figure B.1: Initial states of trajectories generated through optimiser

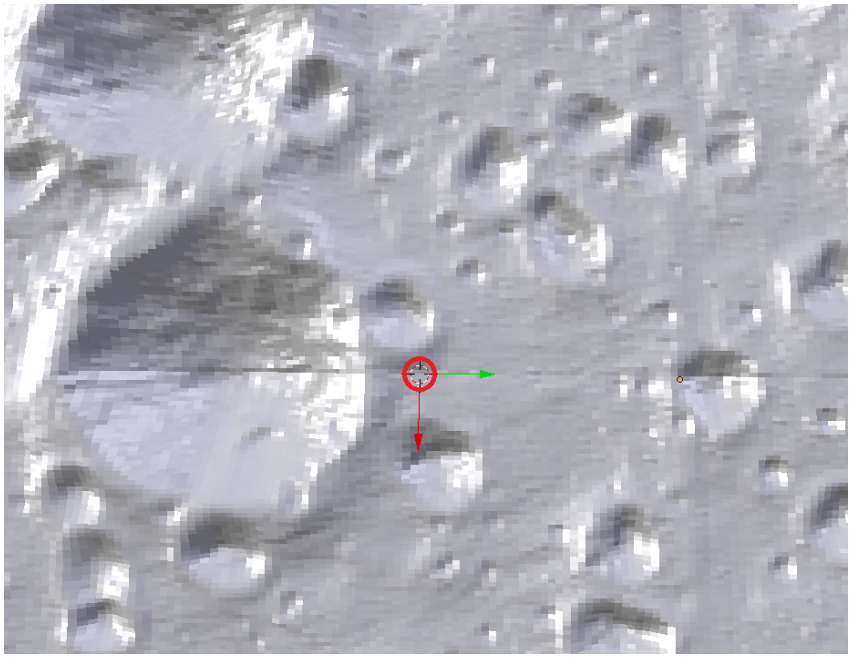


Figure B.2: Position of the landing spot for optimised trajectories in the TRON model (red circle)

was written into an *xml* file with the required format.

To test the transformations, an *xml* file thus generated was used to place the TCP in the lab. The T-MAC position and orientation was measured using the LT. Using camera calibrations with respect to the T-MAC and the TCP, the T-MAC state was first used to obtain the camera's state; and from this one the TCP state was derived (using the camera to TCP calibration). The state of the TCP was transformed from the $\mathscr{F}_{LT}$ frame to the $\mathscr{F}_M$ frame (using the position measurements of the model reflectors). Finally, the state was transformed into the $\mathscr{F}_{KUKA}$ frame with the above mentioned transformation matrices.

As shown in Figure B.3, the calculated state in the $\mathscr{F}_{KUKA}$ frame differs from the commanded one by a translation. This translation was observed to be nearly constant throughout the entire trajectory.

After some crude distance measurements in the lab, it was noted that the initial frame provided – $\mathscr{F}_{PHY}$ – was not the one required – $\mathscr{F}_M$. The initial frame was positioned in the top left corner of the model, rather than at the corresponding reflector. Since the orientation of the two frames are similar, this was corrected with a translation of $\begin{pmatrix} 0.38 & -0.49 & 0.20 \end{pmatrix}^T$.

B.2. Trajectory Sections

Due to the spatial limitations of the lab and of the LT, the following constraints were imposed on the trajectories:

- LT constraints
  - Angle between the Z_{TMAC}-axis and the vector from the LT origin to the T-MAC origin smaller than 45°
  - Distance between the LT origin to the T-MAC origin higher than 1.5 m
- A trajectory confined to the possible movement of the robot in the laboratory
  - $\|x_{KUKA}\| < 1.5$ m
  - $0.2 < z_{KUKA} < 2.5$ m
  - $2 < y_{KUKA} < 11$ m
- A T-MAC position that allows for camera calibration – it is within sight of the LT in the positions used to capture the calibration images

The constraint on the Z_{KUKA}-axis was used as one of the criteria to limit the number of trajectories. The remaining constraints were used in a simple simulation to divide the trajectories into sections. Doing so is important to guarantee the efficient use of the lab, since, in case the LT loses track of the T-MAC during the measurements, the run stops and the software used may crash, thus slowing down the process.

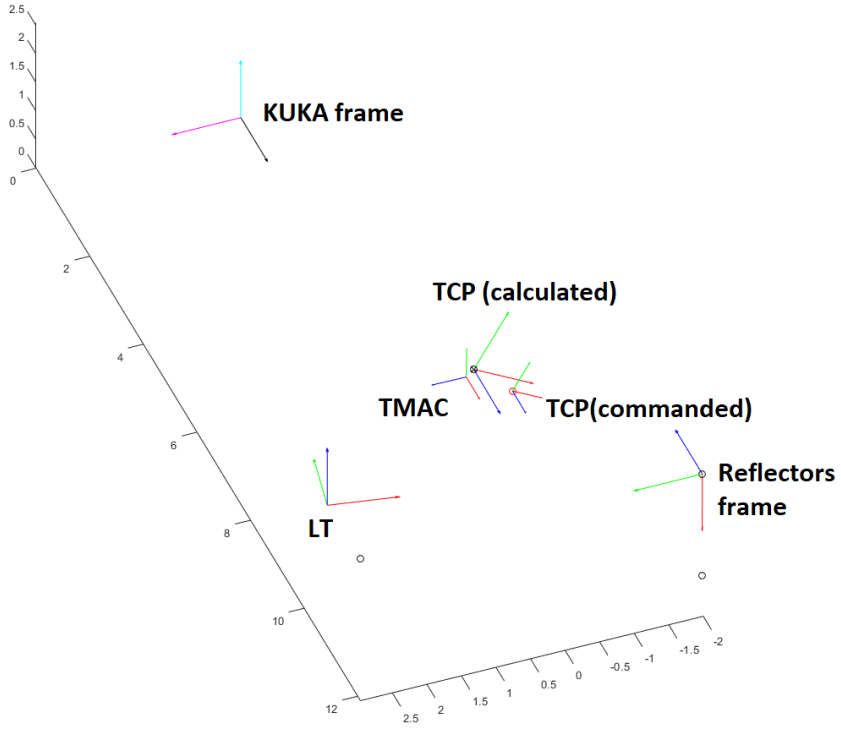


Figure B.3: Comparison between the calculated and the commanded TCP position and orientation in the $\mathcal{F}_{KUKA}$ frame

The position of the LT was simulated by changing its original position in the $\mathcal{F}_{KUKA}$ frame, namely the y_{KUKA}-coordinate. This was done in multiples of 0.6 m, corresponding to the size of the tiles in the lab floor, which constraint the possible positions of the instrument.

For each LT position defined, the angle and distance to each trajectory point was checked. A margin of 15 cm was given for the minimum distance (it was set to 1.65 m instead of 1.5 m) and the angle was calculated with respect to two points located 10 cm at each side of the LT position considered (along the Y_{KUKA}-axis). In case these constraints were satisfied, the point was added to the associated section. Once a point was added to a section, it was no longer considered for the remaining ones (to avoid repeated measurements between sections).

The LT positions were adjusted manually until all possible points were included. A total of three trajectory positions were necessary, noting that three points were excluded.

B.3. Issues in the Lab

Performing the simulation described in Section B.2 greatly improved the work-flow in the lab. However, new issues arose when running the last section (furthest from the model).

To allow positioning the TCP at large values of $x_{KUKA} - \|x_{KUKA}\| > 1.2$m – the base of the robot was commanded (through variable E1) to be closer to y_{KUKA} of the TCP. As a result, the base and arm of the robot would intersect the line of sight between the T-MAC and the LT.

The constraint imposed by the length of the robot's arm was not used to limit the z_{KUKA}-coordinate or to change the position of the robot's base accordingly (as was done with x_{KUKA}). Additionally, since the T-MAC was being measured by a specific part of the LT, the minimum distance of 1.5 m to the origin of the LT was not representative. Instead, this minimum distance was with respect to the specific part measuring the T-MAC orientation.

Finally, the line of sight from the LT (in the last position) to one of the reflectors was intersected by terrain model 1. Thus, the LT was moved about 20 cm towards the centre of the room (along X_{KUKA}). This decreased the distance to the TCP even more.

The points that could not be measured were removed on-site from the trajectory.

Bibliography

Alcantarilla, P. F., Bartoli, A., and Davison, A. J. (2012). "KAZE Features". In Fitzgibbon, A. W., Lazebnik, S., Perona, P., Sato, Y., and Schmid, C., editors, *ECCV (6)*, volume 7577 of *Lecture Notes in Computer Science*, pp. 214–227. Springer.

Alcantarilla, P. F., Nuevo, J., and Bartoli, A. (2013). "Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces". In Burghardt, T., Damen, D., Mayol-Cuevas, W. W., and Mirmehdi, M., editors, *BMVC*. BMVA Press.

Astrium, E. (2006). "navigation for planetary approach and landing-final report.

Bennett, F. and (U.S.), M. S. C. (1972). *Apollo Experience Report: Mission Planning for Lunar Module Descent and Ascent*. Number v. 6846 in Apollo Experience Report: Mission Planning for Lunar Module Descent and Ascent. National Aeronautics and Space Administration.

Carson, J., A. Robertson, E., Pierrottet, D., Roback, V., Trawny, N., L. Devolites, J., J. Hart, J., N. Estes, J., and S. Gaddis, G. (2014a). Preparation and integration of alhat precision landing technology for morpheus flight testing. referenced at introduction as source for stating what alhat is meant for.

Carson, J., A. Robertson, E., Pierrottet, D., Roback, V., Trawny, N., L. Devolites, J., J. Hart, J., N. Estes, J., and S. Gaddis, G. (2014b). Preparation and integration of alhat precision landing technology for morpheus flight testing.

Carson, J., Robertson, E., Trawny, N., and Amzajerdian, F. (2015). Flight testing alhat precision landing technologies integrated onboard the morpheus rocket vehicle.

Cheng, Y. and Ansar, A. (2005). "Landmark Based Position Estimation for Pinpoint Landing on Mars". In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 1573–1578.

Cheng, Y., Goguen, J., Johnson, A., Leger, C., Matthies, L., Martin, M. S., and Willson, R., "The Mars exploration rovers descent image motion estimation system". *IEEE Intelligent Systems*, 19(3):13–21 (2004).

Diebel, J. (2006). Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. Technical report, Stanford University.

Epp, C. D. and Smith, T. B. (2007). "The Autonomous Precision Landing and Hazard Detection and Avoidance Technology (ALHAT)".

Fischler, M. A. and Bolles, R. C., Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395 (1981).

Forsyth, D. A. and Ponce, J. (2003). *Computer Vision: A Modern Approach*. Prentice Hall.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

I. Restrepo, C., Carson, J., Amzajerdian, F., Seubert, C., Lovelace, R., McCarthy, M., Tse, T., Stelling, R., and Collins, S. (2018). Open-loop performance of cobalt precision landing payload on a commercial sub-orbital rocket.

J. Erickson and J.P. Grotzinger (2014). Mission to Mt. Sharp Habitability, Preservation of Organics, and Environmental Transitions. Senior Review Proposal.

Johnson, A., Cheng, Y., Montgomery, J., Trawny, N., Tweddle, B., and X. Zheng, J. (2015). Real-time terrain relative navigation test results from a relevant environment for mars landing.

Johnson, A. and Ivanov, T. (2011). Analysis and testing of a lidar-based approach to terrain relative navigation for precise lunar landing.

Kolb, C. E., Mitchell, D. P., and Hanrahan, P. (1995). "A realistic camera model for computer graphics". In Mair, S. G. and Cook, R., editors, *SIGGRAPH*, pp. 317–324. ACM.

Krüger, H. and Theil, S., Tron-hardware-in-the-loop test facility for lunar descent and landing optical navigation. *IFAC Proceedings Volumes*, 43(15):265–270 (2010).

Krüger, H., Theil, S., Sagliano, M., and Hartkopf, S. (2014). "on-ground testing optical navigation systems for exploration missions". In *9th ESA Conference on Guidance, Navigation & Control Systems*.

Lepetit, V., Moreno-Noguer, F., and Fua, P., Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81(2):155 (2008).

Leutenegger, S., Chli, M., and Siegwart, R. Y. (2011). "BRISK: Binary Robust Invariant Scalable Keypoints". In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pp. 2548–2555, Washington, DC, USA. IEEE Computer Society.

Li, S., Jiang, X., and Tao, T. (2015). Guidance summary and assessment of the chang'e-3 powered descent and landing.

Lingenauber, M., Bodenmüller, T., Bartelsen, J., Maass, B., Krüger, H., Paproth, C., Kuß, S., and Suppa, M. (2013). Rapid modeling of high resolution moon-like terrain models for testing of optical localization methods. In *12th Symposium on Advanced Space Technologies in Robotics and Automation*. European Space Agency.

Liu, J.-J., Yan, W., Li, C.-L., Tan, X., Ren, X., and Mu, L.-L., Reconstructing the landing trajectory of the ce-3 lunar probe by using images from the landing camera. *Research in Astronomy and Astrophysics*, 14(12):1530 (2014).

Liu, Z., Di, K., Peng, M., Wan, W., Liu, B., Li, L., Yu, T., Wang, B., Zhou, J., and Chen, H., "High precision landing site mapping and rover localization for Chang'e-3 mission". *Science China Physics, Mechanics & Astronomy*, 58(1):1–11 (2015).

Mammarella, M., Avilés Rodrigálvarez, M., Pizzichini, A., and María Sánchez Montero, A. (2011). Advanced optical terrain absolute navigation for pinpoint lunar landing.

Mourikis, A. I., Trawny, N., Roumeliotis, S. I., Johnson, A. E., Ansar, A., and Matthies, L., Vision-aided inertial navigation for spacecraft entry, descent, and landing. *IEEE Transactions on Robotics*, 25(2):264–280 (2009).

Muja, M. and Lowe, D. G., "Scalable Nearest Neighbor Algorithms for High Dimensional Data". *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(11):2227–2240 (2014).

Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Oyallon, E. and Rabin, J., "an analysis of the surf method". *Image Processing On Line*, 5:176–218 (2015).

Pierrottet, D., Amzajerdian, F., and Barnes, B. (2014). A long-distance laser altimeter for terrain relative navigation and spacecraft landing. referenced at introduction as introduction to NASA's long range laser altimeter used in ALHAT.

Ribbens, B., Jacobs, V. A., Vuye, C., Buytaert, J., Dirckx, J., and Vanlanduit, S., "high-resolution temporal profilometry using fourier estimation". *Recent advances in topography research*, pp. 61–108 (2013).

Sagliano, M. (2017). Pseudospectral convex optimization for powered descent and landing.

Sagliano, M. (2018). Generalized hp pseudospectral convex programming for powered descent and landing.

Sagliano, M., Theil, S., Bergsma, M., D'Onofrio, V., Whittle, L., and Viavattene, G., On the radau pseudospectral method: theoretical and implementation advances. *CEAS Space Journal*, 9(3):313–331 (2017).

Sakai., S., Sawai, S., Fukuda, S., Sato, E., Kawano, T., Kukita, A., Maru, Y., Miyazawa, Y., Mizuno, T., Nakatsuka, J., Okazaki, S., Saiki, T., Tobe, H., Higuchi, T., Hokamoto, S., Kamata, H., Kitazono, K., Noumi, M., Takadama, K., and Ueno, S. (2015). "Small lunar-lander “SLIM” for the pinpoint landing technology demonstration". In *11th Low Cost Planetary Missions Conference*.

Siddiqi, A. A. (2002). *Deep Space Chronicle: A Chronology of Deep Space and Planetary Probes 1958–2000*, volume 24 of *Monographs in Aerospace History*. NASA SP.

Sidi, M. J. (2006). *Spacecraft dynamics and control: a practical engineering approach*. Cambridge University Press. referenced at rotation calculation from quaternion.

Tchernykh, V., Beck, M., Janschek, K., and Flandin, G. (2006). "An Optical Flow Approach for Precise Visual Navigation of a Planetary Lander". In *Guidance, Navigation and Control Systems*, volume 606 of *ESA Special Publication*, pp. 80.1.

Theil, S., Ammann, N. A., Andert, F., Franz, T., Krüger, H., Lehner, H., Lingenauber, M., Lüdtke, D., Maass, B., Paproth, C., et al. (2017). Aton-autonomous terrain-based optical navigation for exploration missions: Recent flight test results.

Trawny, N., Benito, J., E. Tweddle, B., Bergh, C., Khanoyan, G., Vaughan, G., Zheng, J., Villalpando, C., Cheng, Y., Scharf, D., Fisher, C., Sulzen, P., Montgomery, J., Johnson, A., Aung, M., Regehr, M., Dueri, D., Açıkmeşe, B., Masten, D., and Nietfeld, S. (2015). Flight testing of terrain-relative navigation and large-divert guidance on a vtvl rocket.

Trigo, G. F., Maass, B., Krüger, H., and Theil, S., Hybrid optical navigation by crater detection for lunar pin-point landing: trajectories from helicopter flight tests. *CEAS Space Journal* (2018).

Weng, J., Cohen, P., and Herniou, M., "Camera calibration with distortion models and accuracy evaluation". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10):965–980 (1992). referenced at camera distortions (theory of distortion types).

White, J., Criss, T., and Adams, D. (2009). Aplnav terrain relative navigation helicopter field testing.

Willson, R., Johnson, A., and Goguen, J. (2005). Moc2dimes: A camera simulator for the mars exploration rover descent image motion estimation system. In *'i-SAIRAS 2005'-The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, volume 603.

Yang, X. and Cheng, K.-T. (2012). "LDB: An ultra-fast feature for scalable Augmented Reality on mobile devices". In *ISMAR*, pp. 49–57. IEEE Computer Society.